

PROOF OF CHURCH-ROSSER THEOREM IN CALCULUS OF CONSTRUCTIONS

A thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

by
AROURI NARAYANA

to the
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
APRIL 1991

ACKNOWLEDGEMENTS

First of all, I would like to thank my thesis supervisor Dr.Somenath Biswas for not just teaching the subject but for being a model in all spheres of life.

I thank Durga Ramesh, Peyyeti Sharma and Kordale Rambo for their company and affection which kept me living the life. I won't be able to forget the association of the Telugu Sangha which filled the jollier part of my stay here. Especially, I would like to thank the company of Ravi Chander, DVSSM, Ravi Shaker, Baddu, Bendapudi Perraju, BV Rao and all others.

I would like to thank Kalyan Shanker Basu, Gerard Huet, Christine Pauline Mohring and Mauny for their patience and interest in clarifying my doubts.

CERTIFICATE

This is to certify that the work entitled "PROOF OF CHURCH-ROSSER THEOREM IN CALCULUS OF CONSTRUCTIONS" by Aroori Narayana has been carried out under my supervision and has not been submitted elsewhere for a degree.

S. Biswas

(Somenath Biswas)

Department of Computer Science and Engg'.

Indian Institute of Technology

KANPUR

Kanpur

April 1991.

CSE - 199 -M
11/11-19

11

Acc. No 112199

CSE 1991-M-NAR-PRO

2

or $\frac{1}{2} \text{ of } 1 \text{ plate} = 2$

CONTENTS

Chapter		page
1	INTRODUCTION	1
2	INFORMAL PROOF	12
3	PROOF FORMALISED IN COC	20
4	CONCLUDING REMARKS	55
	REFERENCES	57
	APPENDIX A HOW TO USE COC	59
	APPENDIX B PROOF SCRIPT OF CHURCH ROSSER	

CHAPTER 1

INTRODUCTION

In the current the is we shall use the Calculus of Constructions to provide a constructive proof of Church-Rosser theorem for λ -calculus. The Church-Rosser theorem states that if a λ -term X reduces to λ -terms Y and Z then there exists a λ -term W such that Y and Z reduce to it. The Church-Rosser theorem assures the uniqueness of the normal form. The proof is formalised for λ calculus with de Bruijn notation. The formalised proof script is presented in the appendix.

The λ -calculus was originally conceived by Church as part of a general theory of functions and logic intended as a foundation for mathematics. Although the full system turned out to be inconsistent the subsystem dealing only with functions turned out to be a successful model of computation. Instead of trying to model the logic formulae by pure λ -terms which gave Church an inconsistent system one can model the proof terms of logic by typed λ -terms which gives a consistent system. *Calculus of Constructions* (abbreviated CDC hereafter) [CH88] follows this paradigm and has emerged as an elegant presentation of constructive logic. The proof theoretic strength of the system is not yet completely understood. The inherent structure of the λ calculus for this purpose is higher order impredicative λ -calculus with dependent types.

The CDC is interesting to computer science since it presents the *proofs-as-programs* paradigm in a clear light. When we identify the types with the propositions the resulting proof terms can be seen as the programs which realise their types in Kleene's sense of realisability for intuitionistic logic.

Church-Rosser theorem is a celebrated mathematical result about λ calculus. It implies the consistency of the λ -calculus as a rewrite rule system. It has taken more than thirty years for the evolution of a clear proof of Church-Rosser. Most of the early proofs are either incomplete or defective. The current study presents a machine checking of the proof by Martin-Lof [Lit].

The Martin-Lof's proof for λ -calculus with standard notation is presented in the appendix of [HS72]. In the current study the λ -calculus is formulated in de Bruijn notation. This notation assumes importance in the machine implementations of λ -calculus. The de Bruijn notation does away with the α -reduction. But the proof gets more complex with the properties of de Bruijn indices.

The whole proof as presented in [HS72] is tailored for de Bruijn indices and the proof is machine checked in CDC. The machine checked proof script is presented in the appendix B. The implementation of CDC used for the current study is (CDC V 4.10) developed at INRIA.

A mechanical proof for Church-Rosser in Boyer-Moore logic is presented in [Sh88]. The Boyer-Moore logic is completely different from CDC in content and philosophy. The Boyer-Moore's is a quantifier free first order logic. The language is a form of Pure-Lisp. The terms of Boyer-Moore are either variables or of the form $(FN\ t_1\ \dots\ t_n)$ where FN is an n -ary function and $t_1\ \dots\ t_n$ are terms. The CDC is an higher order impredicative intuitionistic logic.

The following sections of the current chapter present a brief survey of literature on CDC. The second chapter presents an informal proof on the lines of [HS72]. Chapter three is a rigorous formulation of the proof with de Bruijn notation. This chapter can also be seen as an English translation of the machine checked proof script which is added as appendix B. The chapter 4

briefly summarises our experience with the system of COC. Appendix A is an example session of the system V 4.10.

1 COC from λ -calculus point of view

COC can also be viewed in terms of Barendregt's *Generalised Type System (GTS)* [BH90] which clearly shows the place of this formalism amongst such ones. This view also shows that COC is the most general amongst such systems.

In this section we present COC from the λ -calculus point of view. We describe Barendregt's *Generalised Type System (GTS)* as a single uniform presentation of the forest of λ -calculus systems and show the place where the COC nicely fits in GTS.

1.1 Definition (1) (term of GTS) The set T of terms of a GTS is defined by

$$T = V \mid C \mid T T \mid \lambda V. T T \mid \Pi V. T T$$

where C is the set of constants and V is the set of variables.

(ii) (statement) A statement of a GTS is of the form $M : A$ with $A \in T$. M is called the term and A is called its type.

(iii) A context is a finite linearly ordered sequence of statements with distinct variables as terms.

1.2 Definition : A specification of a GTS is a triple (S, A, R) such that

- $S \subseteq C$ the elements of S are called sorts

- A is the set of statements of the form $c : s$ with $c \in C$ and $s \in S$. The elements of A are called axioms.

- R is the set of pairs of the form (s_1, s_2) with s_1 and $s_2 \in S$. The elements of R are called rules.

1.3 Definition (GTS) : Given a specification of a GTS $G = (S, A, R)$ the corresponding λ -calculus system derives the statements relative to a context Γ . The rules $(s1, s2) \in R$ determine which abstractions are allowed.

(Axiom) $\vdash c : s$ if $(c, s) \in A$

(Start)
$$\frac{\Gamma \vdash A : s}{\Gamma \vdash A : A} \text{ if } s \in S \text{ and } x \text{ is fresh}$$

(Weakening)
$$\frac{\Gamma \vdash B : C \quad \Gamma \vdash A : s}{\Gamma, A \vdash B : C} \text{ if } s \in S \text{ and } s \text{ is fresh}$$

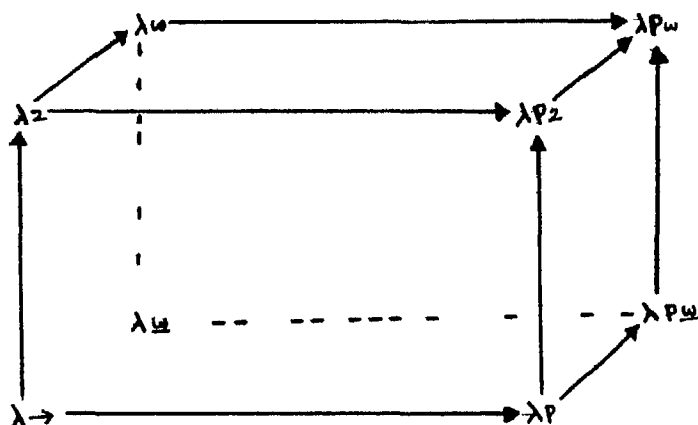
(Π -elimination)
$$\frac{\Gamma \vdash F : (\Pi A. B) \quad \Gamma \vdash a : A}{\Gamma \vdash (F a) : [a/x]B}$$

(Π -formation)
$$\frac{\Gamma \vdash A : s1 \quad \Gamma, x : A \vdash B : s2}{\Gamma \vdash (\Pi A. B) : s2} \text{ if } (s1, s2) \in R$$

(Π -introduction)
$$\frac{\Gamma \vdash A : s1 \quad \Gamma, x : A \vdash B : s2 \quad \Gamma \vdash A : B}{\Gamma \vdash (\lambda x. A) : (\Pi x. A \rightarrow B)} \text{ if } (s1, s2) \in R$$

(Conversion)
$$\frac{\Gamma \vdash A : B \quad \Gamma \vdash B : s}{\Gamma \vdash A : B} \text{ if } s \in S \text{ and } B \equiv_{\beta} B$$

For Various GTS specifications we get different λ -calculus systems. The calculus of constructions is a λ -calculus system with sorts $S = * \square$ and axioms $A = (* \square \square)$ and rules $R = (* *)$ $(\square \square)$ $(\square *)$ $(\square \square \square)$. The rule $(*) \in R$ gives the system the elements dependent on elements $(\square *) \in R$ gives the polymorphic types $(* \square) \in R$ gives the types dependent on elements and $(\square \square) \in R$ gives the types dependent on types. The CDC is thus an impredicative higher order typed λ -calculus with dependent types. Without the rule $(*) \in R$ we get the system F_ω of Girard. Thus the CDC is a conservative extension of system F_ω . The relation of CDC with seven other common λ -calculus systems can be shown as a cube called λ -cube.



λ -cube

For all the eight systems the sorts and axioms are common.

1. $e \vdash * \square$ and $A = (* \square)$ The rules are as follows

- (1) λ (simply typed λ calculus) $R = (* *)$
- (2) $\lambda 2$ (System F) $R = (* *) (\square *)$
- (3) $\lambda \omega$ $R = (* *) (\square \square)$
- (4) $\lambda \omega$ (System F ω) $R = (* *) (\square *) (\square \square)$
- (5) λP (LF) $R = (* *) (* \square)$
- (6) $\lambda P 2$ $R = ((* *) (\square *) (* \square))$
- (7) $\lambda P \omega$ $R = (* *) (\square \square) (* \square)$
- (8) $\lambda P \omega$ (COC) $R = (* *) (\square *) (\square \square) (* \square)$

Thus the COC or $\lambda P \omega$ is the most general among the possible systems. The system λP is given that name because predicate logic is interpreted in it. The approach followed for this interpretation is called *propositions-as-types* paradigm and is the basis of several languages like AUTOMATH [dB80], NUPRL [Co56], LF [CH87].

In all the above systems the terms of $*$ are called *types* and the terms of \square are called *kinds*.

2 COC from logic point of view

In the previous section we have seen that COC is an impredicative higher-order λ calculus with dependent types. By identifying $*$ with Prop, the set of all logic propositions, we get an impredicative higher order minimal intuitionistic logic. This is the pure Calculus of Construction as presented in [CH88]. Here we try to model the syntactic behavior of logic by λ calculus β -reduction. In this framework the types are identified with the formulae and the λ -terms with the proofs. Instead of directly identifying the types with the propositions, another way of viewing this is the type corresponding to a proposition is the type of proofs of the proposition. The logic thus becomes the expressions of the fact that a proposition is true if and only

if its corresponding type of proof is inhabited

In [Coq89] Coquand call the types of previous section a small types and kinds of previous section as big type. Only the small types correspond to the formulae of logic. The big types are the types of formulae and higher order predicates. Both the small and big types can be used as individual types. We have both the calculus of functions (when we consider types as individual types) and calculus of logic (when we consider the types as the types of proofs) in a single framework. Of course there is some confusion of understanding between the two.

We have terms of 4 levels. The terms of a level are the types of the terms of a lower level.

The terms of the 0th level are proof terms. The terms of level 1 are the small types or the formulae of logic. Only these are called types by Barendregt[Bar90]. The terms of level 2 are the type of terms of level 1 and are called big types by Coquand[Coq89] and kinds by Barendregt[Bar90]. Here we are able to form $A \rightarrow Prop$, $Prop \rightarrow Prop$ etc. The terms of level 3 are singleton. This is \Box of [Bar90] and Type of [Coq89]. This is the type of all types.

We are able to form four types of products:

(i) Product formation from small types to small types give rise to the concept of elements depending on elements. eg. If A and B are of type $*$ the type $A \rightarrow B$ is also of type $*$. This is arrow type in λ -calculus and implication in logic.

(ii) The product formation from big types to small types give rise to the effect of polymorphism of the λ calculus and second order quantification of logic. For eg. $A \rightarrow \lambda x. (A \rightarrow *) \rightarrow (A \rightarrow A) : *$. The type $(A \rightarrow *) \rightarrow (A \rightarrow A)$ is the type of polymorphic identity in λ -calculus. When we identify $*$ with $Prop$ in logic we have the formula $(A \rightarrow Prop) \rightarrow (A \rightarrow A)$ of type $Prop$. It is here we get an

impredicative system. The concept of defining an element of a set by using the description of the same set is called *impredicativism*. Here we are defining an element of Prop by quantifying over Prop itself. The actual implementation of CDC V4.10 done by Huet [H89] contains an infinite number of hierarchies. All the levels beyond level one are predicative while the level one is impredicative. The impredicativity at two levels gives rise to inconsistency to the system as has been observed by Girard [G86] for system U.

(iii) The product formation from small type to a big type gives the effect of element dependent types. For eg $A * 1 - (:) A * \square$ which is the type $A \rightarrow \text{Prop}$ the type of predicates over A .

(iv) The product formation from a big type to big type gives the ability to use higher order quantification. For eg $A * B * 1 - \lambda(A \text{ Prop}) \lambda(B \text{ Prop}) (C \text{ Prop}) (A \rightarrow B \rightarrow C) \rightarrow C$. This is the connective AND which is of the type $\text{Prop} \rightarrow \text{Prop} \rightarrow \text{Prop}$. The connective AND is generic in the sense that we can plug in any two propositions.

3 Implementation of CDC

The implementation of CDC which is used in the current study is the version 4.10 written by Gerard Huet [H89] developed at INRIA France. This section briefly describes the proof description language of this implementation called Mathematical vernacular developed by Gilles Dowek [D89] the facility for inductive definitions developed by Pauline [P89] and an automatic theorem prover called Synthesis developed by Coquand [C89S].

3.1 Mathematical Vernacular :

The purpose of the proof description language is to make the language of CDC as close as possible to the language in which the mathematical books are written. Each and every vernacular

tatement is translated into the basic commands of the constructive engine [H89] of LDC. The fundamental vernacular which is mainly made use of in the machine proof of Church-Rosser is explained below.

In vernacular a variable is declared by

Parameter <name of variable> <type>

The syntax for adding an axiom is

Axiom <name> <statement>

The syntax for proving a new theorem is

Theorem <name of theorem> <statement>

Proof proof of the theorem

New definition is added by the syntax

Definition <name>=<body>

Local variables and hypotheses are added respectively by

Variable <name> <type> and

Hypothesis <name> <statement>

There is a provision for splitting the proof into *Sections*, *Subsections* and *Chapters* with the usual scoping rules.

3.2 Inductive definitions :

Inductive definitions are the means of giving complete specification of a notion from the description of their constructors. A macro command is available to the user which generates the type introduction and elimination rules from the specification. For eg. the definition of sum type is given as

Inductive Definition sum [A B Prop] Prop

= left A->(sum A B)

| right B->(sum A B)

It generates the following

j) = (C *) (A -> C) (B -> C) -> C

= [H:(sum A B)]H

```

left = (Ca AJEC *JECf:A- CJCJ B- CJC(f a))    A (sum A B)
right = (Cb BJEC *JECf A- CJCJ B CJC(j b))    B- (sum A B)

```

Two specifications in two different constructors introduce a sum type between them and in the same constructor introduce a product type between them

3.3 Tactics

The tactics theorem prover called Synthesis machine is developed by Coquand [Cq89a]. The philosophy of the tactics is the goal directed proving which is the reverse of the natural deduction. The tactic is a CAML function which when applied to a goal reduces it to simpler subgoal. A tactical is a higher order CAML function for combining tactics to build new tactics. Different tactics are provided for different inference rules of COC. The tactic which were found to be much useful in the construction of the proof of Church Rosser are

(a) assumption tactic : This takes the proof of a goal from the hypotheses context. This tactic corresponds to rule *start*.

(b) intro tactic : This introduces the abstraction in the proof. This corresponds to Π -*introduction* rule.

(c) resolve tactic : This introduces an application in the proof structure. The effect corresponds to Π -*elimination* rule.

(d) change tactic : The effect of this tactic corresponds to *conversion* rule.

The basic tacticals which were found to be useful are

tac1 THEN tac2

This applies the `tac2` tactic to all the new goals generated by the `tac1` tactic

REPEAT tac

This repeats the tactic `tac` till it fails

A compound tactic called *automatic* built from the above tactics was found to be very much useful. This performs a series of five successful resolutions with the lemmas added to `add_resolve_list` interspersed with assumptions and fails if it can not complete the current goal

CHAPTER 2

INFORMAL PROOF

1 Introduction to λ -calculus and de Bruijn notation

The λ -calculus is a type free theory about the functions as rule rather than graphs. It studies the problems such as what all can be computed and which computable processes are equivalent. These computable processes are called λ terms of the λ -calculus. In our presentation the main relation between the λ -terms is reduction which is sufficient for the proof of Church-Rosser.

We begin by assuming that there is an infinite set V of variable and a finite or infinite set C of constants. An atom is a constant or a variable.

1.1 Definition (λ -terms) The set of λ -terms Λ is defined by

- (i) Every atom is a λ term
- (ii) If X and Y are λ -terms then $(X Y)$ is a λ -term
- (iii) If Y is a term and x is a variable then $\lambda x. Y$ is a λ -term

Examples of λ -terms are $(\lambda x. (\lambda y. (x y)))$ $((\lambda x. x)(\lambda y. (x y)))$

1.2 Definition (Free and Bound variables) An occurrence of a variable x in Y is bound iff it is inside a part of Y of the form $\lambda x. \dots$ otherwise it is free

Substitution is the process of replacing all the free occurrences of a variable in a term by another term.

1.3 Definition (Substitution) For any terms M , N and any variable x the result $[N/x]M$ of substituting N for every free occurrence

of \langle in M is defined as follows by induction on the construction of M

(i) $\text{CN/} \langle \rangle = N$

(ii) $\text{CN/} \langle \lambda y \cdot \dots \rangle = y$ for all atoms $y \neq x$

(iii) $\text{CN/} \langle \lambda M_1 M_2 \rangle = ((\text{CN/} \langle M_1 \rangle)(\text{CN/} \langle M_2 \rangle))$

(iv) $\text{CN/} \langle \lambda y M \rangle = \lambda y M$ if $y \neq x$
 $= \lambda y \text{CN/} \langle M \rangle$ otherwise

To ensure that the substitution has the intended meaning no free variable occurrence in N can be allowed to become bound in $\text{CN/} \langle M \rangle$. For this we rename the bound variable x as and when needed. Two terms which are equivalent except for the renaming of bound variables are called α -reducible to each other.

1.4 Definition (α -reduction) A term X goes to a term Y in a α -step iff X and Y are identical except that the names used to denote the occurrence of variables bound by a λ in X and the corresponding in Y might differ uniformly.

The computation is actually done by a rule known as β -step. A subterm of the form $(\lambda x. M)N$ is called a redex.

1.5 Definition (reduction) A term X goes to Y in a β -step iff a subterm $(\lambda x. M)N$ of X is replaced by $\text{CN/} \langle M \rangle$ to obtain the term Y . A term X reduces to a term Y iff Y is obtained from X by a finite (perhaps empty) series of β -steps and α -steps. This is denoted as $X \rightarrow^* Y$.

Hence β -reduce relation is the reflexive transitive closure of β -step relation.

eg $(\lambda x. (\lambda y. (y \ x))) v \rightarrow^* \text{CN/} \langle \lambda y. (y \ x) \rangle = (\lambda y. (y \ v))$

1.6 de Bruijn Notation

With the previous notation for λ -terms we have to take care

of renaming of bound variable in the reduction process at every β -step. With the notation developed by de Bruijn in [dB72] for the implementation of Automath two terms which are α equal are always syntactically identical. In this notation a variable occurrence in a λ -term is represented by an integer denoting the depth from the binding λ . We also do away with the tagging of name of variable with λ in the abstraction. If there are n λ s during the firing between the binding λ and the occurrence of variable that place of variable is represented by $n+1$.

For eg $(\lambda (\lambda y () (\lambda z)))$ is represented in de Bruijn notation as $(\lambda (\lambda (2 (\lambda 1))))$

If corresponding to a variable occurrence represented by n in a subterm M there is no n th closest surrounding λ then that variable occurrence is free in M . For eg the underlined variable occurrences are free in $((\lambda (\underline{2} (\lambda (2 \underline{3})))) (\lambda (\underline{4} 1)))$

During the β -reduction of the redex $(M N)$ the free variables of both the M and N are affected. Consider $M = (\lambda (\underline{2} (\lambda (\lambda 3))))$ and $N = (\lambda \underline{2})$ the underlined variable occurrences being free. Consider $(M N)$ is the subterm of $X = (\lambda (M N))$. The free variables of M and N are referencing the outermost λ in X . When we reduce the redex $(M N)$ in a naive fashion we get $X = \lambda (\underline{2} (\lambda (\lambda (\lambda \underline{2}))))$. This is wrong since now the free variables in M and N are not referencing the outermost λ in X . The free variables of M and N have to be properly modified. All the free variables of M have to be decremented by one since a λ is being destroyed in M . All the free variables of N have to be incremented by one as we traverse a λ recursively in M during the substitution since an extra λ is going to be added to the front of N . So the correct result is $X = \lambda (1 (\lambda (\lambda (\lambda 4))))$

Since variables do not have names there is no need for renaming bound variables during the substitution. The de Bruijn

notation assumes importance in the computer implementation of λ -calculus and functional programming languages. One distinction with the standard notation is that if M is the subterm of X the order in which the free variable occurrences in M become bound in X is fixed regardless of X . So it is helpful to think of terms as always being subterms of some other bigger terms.

The λ -term with the de Bruijn notation can be formally defined as follows:

1.7 Definition (λ -terms with the de Bruijn notation) The set Λ of λ -terms is defined by

$$\Lambda = \bigcup_{n \geq 0} \Lambda^n$$

where Λ^n are the sets generated by the inductive rules

$k \in \Lambda^n$ if $1 \leq k \leq n$ (variables)

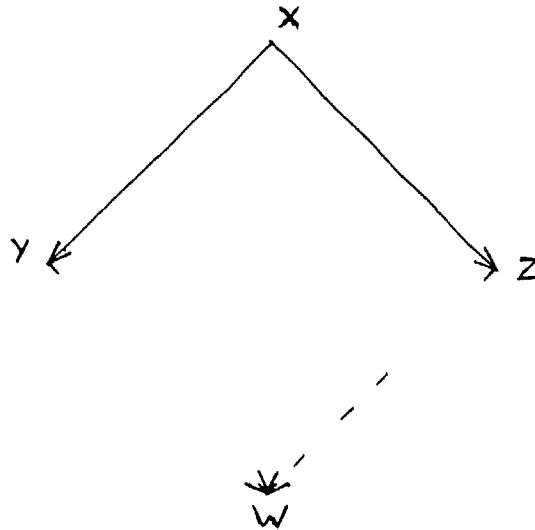
$\lambda N \in \Lambda^n$ if $N \in \Lambda^{n+1}$ (abstraction)

$(MN) \in \Lambda^n$ if $M, N \in \Lambda^n$ (application)

The closed terms are the terms in Λ^0 . A detailed discussion of λ -calculus with this notation can be found in the appendix of [Ba80] and in [dB72].

2 Church-Rosser Theorem

For all $X Y Z$ if $X \twoheadrightarrow Y$ and $X \twoheadrightarrow Z$
then there exists a W such that $Y \twoheadrightarrow W$ and $Z \twoheadrightarrow W$



The statement of the theorem can be shown as in the above figure and that is why it is also called diamond lemma. The term X is called the source of the diamond. Another way of stating this theorem is

All λ -terms can be the sources of diamonds

The Church-Rosser theorem assures the uniqueness of the normal form i.e. all reduction paths do terminate at the same term if at all they terminate.

3 Walks relation

In this section we introduce another relation between two λ -terms called *walks* relation. The proof of Church-Rosser consists in finding a simple relation which has diamond property such that reduction relation is the transitive closure of this new relation. The first trial would be to consider β -step as this new relation since we already know that reduction is the transitive closure of

β -step. But a single β -step relation does not have diamond property. A counter example can illustrate this case. Consider $X = ((\lambda x. ((\lambda y. y) x)) ((\lambda y. y) a))$ and $Z = (((\lambda y. y) a) ((\lambda y. y) a))$. We have that X reduces to Y and Z in a single β -step. But now we can not find a W such that Y and Z reduces to it in a single β -step.

Martin-Löf and Tait consider a relation consisting of a number of β -steps such that inner redexes are always contracted before the outer redexes. They call it *special reduction* and [ST88] calls it *walks*. It is observed that this special reduction has the diamond property and that usual reduction is the transitive closure of this relation.

3.1 Definition (Residual)

Let U be a term containing β -redexes P, Q such that Q does not contain P . Let U' be the result of contracting Q in U . If P, Q are non-overlapping the residual of P in U' is P itself. If $P=Q$ there will not be any residual of P in U' . If Q is a part of P and $P = (\lambda y. M)N$. Then contraction of Q leaves $(\lambda y. M')N$ or $(\lambda y. M)N'$ which is the residual of P in U' .

3.2 Definition (Special reduction or walk)

Consider R_1, \dots, R_n as redexes of X . Contract any R_i which contains no other R_j . This leaves $n-1$ residuals $R_1, \dots, R_{i-1}, R_{i+1}, \dots, R_n$. Contract any R_j which contains no other R_k . This process is continued till no residuals are left. Then X is said to specially reduce or walk to the resulting Y .

4 The main proof and lemmas involved

In this section we give an informal proof of Church-Rosser for standard notation of λ -calculus. The proof for de Bruijn

notation needs few more lemmas regarding the incrementing of free variables and substitution. A rigorous proof with this notation is given in the next chapter which is nothing but English translation of machine checked proof. The proof involves three steps: (i) Showing that walks follow diamond property, (ii) showing that the reflexive transitive closure of walks (called n-walk) follows diamond property, (iii) Showing that diamond property of n-walks implies the diamond property of reduce relation.

Step (i) The statement we need to prove is: *If X walks to Y and Z then there exists a W such that Y and Z walk to it.* The proof is by induction on the structure of X.

Case 1 (X is an atom) By definition Y and Z are also the same atom X. Choosing $W=X$ we have Y and Z walk to W.

Case 2 ($X=\lambda x.M$) By definition there exist M_1, M_2 such that $Y=\lambda x.M_1$, $Z=\lambda x.M_2$ and M_1 walks to M_2 and M . From the induction hypothesis for M we have M' such that M_1 and M_2 walk to it. Choosing the $W=\lambda x.M'$ will satisfy the requirements.

Case 3 ($X=(M N)$) There arise four cases depending on whether X is a redex and if so whether it is the last redex to be β reduced in the walk to X and Y. We consider here the case of X being a redex and in both the walks to Y and Z it is the last redex to be β -reduced. The other three cases are dealt in the rigorous proof of next chapter.

We have $X=(\lambda x.A) B$, $Y=(\lambda x.A_1) B_1$, $Z=(\lambda x.A_2) B_2$ such that A and B walk to A_1 and B_1 and A and B walk to A_2 and B_2 . By the induction hypothesis for A and B we have A' and B' such that A_1 and A_2 walk to A' and B_1 and B_2 walk to B' . Choosing $(\lambda x.A') B'$ will satisfy our requirements given the following lemma.

Lemma (Substitutivity of walks): If Ay walks to Aw and Ny walks to Nw then $ENy/\exists Ay$ walks to $ENw/\exists Aw$. The proof of the lemma is by induction on the structure of Ay and needs the following lemma.

Lemma (subst-subst): $EN/\exists C/\forall yM = E(EN/\exists C)/yJ(EN/\exists M)$

The proof of this lemma is by induction on the structure of the term M . With the de Bruijn notation the proof of this lemma demands few more lemmas like *incr-subst*, *incr-incr* etc. which are explained and proved in the next chapter. The machine proof of the main theorem is contained in the Chapter walk diamond of machine proof.

Step (ii) The proof that n -walks the reflexive and transitive closure of walks follows diamond property involves the induction on the length of walk chain. This proof is shown in the machine proof as Chapter transitivity.

Step (iii) The proof that diamond property of n -walks implies the diamond property of reduce relation consists of showing that a single walk can be considered as a series of β -steps and that a β -step can be considered as a single walk step. The proof of these results are contained in Chapter c-r of machine proof.

CHAPTER 3

PROOF FORMALISED IN COC

In the previous chapter the proof of the Church-Rosser for standard notation is presented in an informal manner. In this Chapter the formal and rigorous proof for Church-Rosser is presented. The proof of this chapter is intended to be more or less an English translation of machine checked proof which is added as appendix B to this thesis. That is why the proof contains somewhat different notation from the standard literature. For example a statement $\vdash C_1 / C_2$ of standard literature is written as $\text{subst}(C_1, C_2)$ in the current Chapter which is actually represented in COC as $(\text{subst } C_1 \ C_2)$. Throughout the chapter the variables which are not explicitly bound in the statements of the lemmas and theorems are understood to be universally quantified.

A constructive proof of Church-Rosser formalised in Boyer-Moore logic is presented in [SF88]. We have followed this work closely to provide our proof in COC.

The section 1 give the basic formulation of λ -calculus as has been done for the machine proof. The section 2 gives some lemmas regarding the operations of incr_free_var and subst . Incr_free_var is the operation of incrementing the de Bruijn indices of the free variables of a term and subst is the operation of substitution of a term in another term for some variable. Most of the lemmas of this section are not needed for standard notation of λ calculus. Hence these are absent in the standard literature. The section 3 presents the lemma of substitutivity of walks. The section 4 contains the proof that walk allows the diamond property. The section 5 gives the proof that reflexive transitive closure of walks relation allows the diamond property. The section 6 gives the proof of the Church-Rosser.

1 Formulation of the basic theory of λ -calculus :

In this section the basic theory of λ -calculus which is needed for the proof of Church-Rosser is presented. We define the λ -terms and subterms. The operations of incrementing the free variables and substitution are explained. Then we define the β -step and ω -step relations and their reflexive transitive closure. All the definitions closely resemble their counterpart in the machine proof. For example the application of the predicate *subst term- nat- term- prop to* ; *y z and w* is represented as *subst(y ~ w)* while in the machine proof it is represented as *(subst y w)*. This is done for the readability of the proof.

1.1 Definition λ -terms :

A type *const* of constants is assumed to exist. The type *nat* of natural numbers is defined inductively by the constructors *zero* and *successor*. The type *term* of λ -terms is defined in the machine proof as

Inductive Definition term Data =

constant const->term

/variable nat->term

/lambda term->term

/apply term->term->term

For an element *c* of type *const* *constant(c)* is a term. For a natural number *v* *variable(v)* is a term. For terms *t1* and *t2* *lambda(t1)* and *apply(t1 t2)* are terms. The symbol *Pred* stands for the predecessor function.

From here onwards the exact syntax of the definitions as in the machine proof is not given. Only an English equivalent of the definitions is given. The machine proof is presented in the appendix B. We use the same names for predicates, types, functions and theorems as has been done in machine proof so that the

machine proof script is understandable

1 2 Definition Sub_term

The formula $\text{sub_term}(x\ y)$ means that x is the sub_term of y . The predicate $\text{sub_term term} \rightarrow \text{term} \rightarrow \text{prop}$ is defined as follows

- (1) For all terms x $\text{sub_term}(x\ x)$
- (ii) $\text{sub_term}(x\ m)$ iff $\text{sub_term}(x\ \text{landa}(\pi))$
- (iii) either $\text{sub_term}(x\ \pi_1)$ or $\text{sub_term}(x\ m_2)$
iff $\text{sub_term}(x\ \text{apply}(\pi_1\ \pi_2))$

1 3 Definition (Incr_free_var) :

With de Bruijn notation in the β -reduction of $(\lambda\ X)Y$ the free variables of Y are affected. For example we have $X \rightarrow \lambda\ 2$ and $Y = 1$. The term Y has a free variable which is bound by the nearest surrounding λ . If we substitute Y in X for the first bound variable in a naive fashion the β -reduction gives us $\lambda\ 1$. But this is wrong since substitution resulted in a term which does not have any more free variables. The free variable of Y got bounded in X during the substitution. The correct result of substitution is $\lambda\ 2$. So during the substitution the free variables of Y have to be incremented by one each time we recurs down a λ in the structure of X .

The operation of incrementing the free variables of Y is defined by induction on the structure of Y . This operation can be defined as a two place function with the arguments the term and a counter to keep track of the number of λ s surrounding the subterm under consideration. The formula $\text{incr_free_var}(i\ y)$ means that the free variables of x are the variables which are bound beyond i levels and these variables are incremented by one to get the term y . The initial value of the counter has to be zero.

The predicate incr_free_var of the type $\text{term} \rightarrow \text{nat} \rightarrow \text{term} \rightarrow \text{Prop}$ is defined recursively on the structure of

the term whose free variables are being incremented is follows

- (a) If $y = \text{constant}(c)$
then $\text{incr_free_var}(i, y)$ iff $y = \text{constant}(c)$
- (b) If $y = \text{variable}(v)$ and $i < v$
then $\text{incr_free_var}(i, y)$ iff $y = \text{variable}(S(v))$
- (c) If $y = \text{variable}(v)$ and $i \geq v$
then $\text{incr_free_var}(i, y)$ iff $y = \text{variable}(v)$
- (d) If $y = \text{lambda}(a)$
then $\text{incr_free_var}(i, y)$ iff
 $y = \text{lambda}(a)$ and $\text{incr_free_var}(a, S(i), a)$
- (e) If $y = \text{apply}(m1, n1)$
then $\text{incr_free_var}(i, y)$ iff $y = \text{apply}(m2, n2)$
 $\text{incr_free_var}(m1, i, m2)$ $\text{incr_free_var}(n1, i, n2)$

1.4 Definition Sbst :

During the β -reduction of $(\lambda X)Y$ the free variables of X are also affected since the λ in X is lost in course of β -reduction. This would mean that there is one fewer λ surrounding the free variables of X . For example consider $X = \lambda (3 (2 1))$. Here 1 is the bound variable X , 2 is the variable for replacement during the β -reduction and 3 is the free variable. If we reduce in a naive fashion we get $\lambda (3 (Y 1))$. But the correct result is $\lambda (2 (Y 1))$. The free variable 3 has to be decremented by one since a surrounding λ is lost. During the recursive definition of substitution a counter is maintained to recognise the free variable occurrences.

The formula $\text{sbst}(y, w)$ means that y is the variable to be replaced, all the variables less than y are bound in w which are to be kept as they are, all the variables greater than y are free which are to be decremented and w is the result of substituting for the occurrence of y in w . The predicate sbst of type $\text{term nat term} \rightarrow \text{prop}$ is defined recursively on the

structure of the term in which the substitution is being performed as follows :

- (a) If $\text{is_constant}(c)$ then
 $\text{subst}(y, w) \text{ iff } w = \text{constant}(c)$
- (b) If $\text{is_variable}(v)$ and $y = v$ then
 $\text{subst}(y, w) \text{ iff } w = \text{variable}(\text{Pred}(v))$
- (c) If $\text{is_variable}(v)$ and $v \neq y$ then
 $\text{subst}(y, w) \text{ iff } w = \text{variable}(v)$
- (d) If $\text{is_variable}(v)$ and $y = v$ then
 $\text{subst}(y, w) \text{ iff } w = y$
- (e) If $\text{is_lamda}(a)$ then
 $\text{subst}(y, w) \text{ iff } w = \text{lamda}(a) \text{ incr_free_var}(0, y)$
 and $\text{subst}(y, w) a a$
- (f) If $\text{is_apply}(m1, n1)$ then
 $\text{subst}(y, w) \text{ iff } w = \text{apply}(m2, n2) \text{ subst}(y, m1, m2) \text{ and } \text{subst}(y, n1, n2)$

1.5 Definition b_step :

The formula $b_step(y)$ means that y is got by β -reducing x in a single step. The predicate b_step of the type $\text{term} \rightarrow \text{prop}$ is defined recursively on the structure of the term which is being β -reduced as follows

- (a) If $\text{is_lamda}(m)$ then
 $b_step(y) \text{ iff } y = \text{lamda}(m) \text{ and } b_step(m, m)$
- (b) If $\text{is_apply}(m, n)$ then
 $b_step(y) \text{ iff either}$
 - (i) $y = \text{apply}(m1, n)$ and $b_step(n, m1)$
 - or
 - (ii) $y = \text{apply}(m, n1)$ and $b_step(n, n1)$
 - or
 - (iii) $m = \text{lamda}(a)$ and $\text{subst}(n, one, a, y)$

1.6 Definition n_b_step

The relation n_b_step is the reflexive and transitive closure of relation b_step . The formula $n_b_step(x, y, n)$ means that x b -reduces to y in n b -steps. The predicate n_b_step of the type $term - term - nat \rightarrow prop$ is recursively defined on the length of the b_step chain as follows:

(a) Reflexivity over a single b_step

$n_b_step(x, x, 0)$ iff $x = x$

(b) Transitivity over a single b_step

$n_b_step(x, z, S(n))$ iff $n_b_step(x, y, n)$ and $b_step(y, z)$

1.7 Definition reduce :

A term x is said to reduce to y iff there exists a number n such that $n_b_step(x, y, n)$ holds.

1.8 Definition Walks :

The motivation for defining a special reduction walk has been explained in section 3 of Chapter 2. The proof of Church-Rosser consists in showing that the reflexive transitive closure of walk relation allows diamond property and in showing that a b -step can be seen as a walk and a walk can be seen as a series of b -steps. The formula $walks(x, y)$ means that x walks to y or in other words x transforms to y in a finite number of b reductions performed in an inside first manner.

The relation $walks$ of type $term - term \rightarrow prop$ is defined recursively on the structure of the term which is being walked from as follows:

(a) If $x = constant(c)$ then

$walks(x, y)$ iff $y = constant(c)$

(b) If $x = variable(v)$ then

$walks(x, y)$ iff $y = variable(v)$

(c) If $x = \text{lamda}(m)$ then
 $\text{walks}(x, y)$ iff $y = \text{lamda}(m1)$ and $\text{walks}(m, m1)$

(d) If $x = \text{apply}(m1, n1)$ then
 $\text{walk}(x, y)$ iff either
 (i) $y = \text{apply}(m2, n2)$ $\text{walks}(m1, m2)$ and $\text{walks}(n1, n2)$
 or
 (ii) $x = \text{lamda}(a1)$ $m2 = \text{lamda}(a2)$ $\text{walks}(m1, m2)$
 $\text{walks}(n1, n2)$ and $\text{subst}(n2 \text{ one } a2, y)$

1.9 Definition n_walks :

n_walks is the reflexive transitive closure of $walks$ relation. The formula $n_walks(x, y, n)$ means that $walks$ to y in n walk steps. The relation n_walks is defined recursively on the length of the walk chain as follows

- (a) Reflexivity over walks relation
 $n_walks(x, x, 0)$ iff $x = x$
- (b) Transitivity over walks relation
 $n_walks(x, w, S(n))$ iff $n_walks(x, v, n)$ and $walks(v, w)$

1.10 Induction schemas

We have following two schemas of induction on the structure of term

Axiom $\text{sub_term_induction}$ $(\forall x: \text{term})(\forall P: \text{term} \rightarrow \text{Prop})$
 $((\forall c: \text{const}) P(\text{constant}(c))) \rightarrow$
 $((\forall v: \text{nat}) P(\text{variable}(v))) \rightarrow$
 $((\forall m: \text{term})(\exists n: \text{term}) \text{sub_term}(m, n) \rightarrow P(n) \rightarrow$
 $P(\text{lamda}(m))) \rightarrow$
 $((\forall m1: \text{term})(\forall m2: \text{term})$
 $((\forall n: \text{term}) \text{sub_term}(m1, n) \rightarrow P(n))$
 $((\forall y: \text{term}) \text{sub_term}(y, m2) \rightarrow P(y)) \rightarrow$
 $P(\text{apply}(m1, m2))) \rightarrow$
 $P(x)$


```

Axiom term_induction : (∀ t:term)(∀ P:term → Prop)
  ((∀ c:const) P(constant(c)))
  ((∀ v:nat) P(variable(v))) →
  ((∀ π:term) P(π) → P(lam1(π)))
  ((∀ π1:term)(π2:term) P(π1) → P(π2) →
    P(apply(π1 π2))) →
  P(t)

```

The schema `sub_term_induction` is the stronger of the two in the sense that `term_induction` can be derived from it and not vice versa.

2 Some properties of `incr_free_var` and `subst` :

This section presents some properties of `incr_free_var` and `subst`. In all the lemmas the cases which are stated to be *trivial* actually result in absurdities. Anything can be proved from absurdity. Hence the proofs are trivial.

(a) Lemma `subst_not_free_in` :

This lemma says that if we increment the free variables of a term bound beyond `i` levels then the term is free for the variable of `S(i)`th level. The statement of the lemma is :

$$\text{incr_free_var}(y \ 1 \ y) \rightarrow \text{subst}(x \ (S \ 1) \ y \ w) \rightarrow (w=y)$$

Proof : The proof is by induction on the term `y`.

We have `incr_free_var(y 1 y)` (1)

`subst(S(1) y w)` (2)

We have to prove `w=y`.

Variable case: We have `y=variable(v)` (3)

For `v 1` : The variable(`v`) in the eq (1) is bound so from the

definition of incr_free_var $y = \text{variable}(v)$ This variable is once again bound in the eq (2) so $w = \text{variable}(v)$

For $v = 1$: The $\text{variable}(v)$ is bound in the eq (1) so $y = \text{variable}(v)$ Once again this variable is bound in the eq (2) hence $w = \text{variable}(v)$

For $v \neq 1$: The $\text{variable}(v)$ is free in the eq (1) so $y = \text{variable}(v+1)$ Once again this variable is free in the eq (2) so from the definition of bst $w = \text{variable}(v)$

Lambda case We have $y = \text{lamda}(n)$ (4)

From the definition of incr_free_var for lamda case and eq (1) there exists n such that

$$y = \text{lamda}(n) \quad \text{incr_free_var}(n \text{ S}(1) n) \quad (5)$$

From the definition of subst and eq (2) there exist x and n such that

$$\text{incr_free_var}(x \text{ } 0) \quad w = \text{lamda}(n) \quad \text{subst}(S(S(1)) n n) \quad (6)$$

From the induction hypothesis and eq (5)&(6) we have $n = n$ Hence $w = \text{lamda}(n)$

Apply case: We have $y = \text{apply}(n1 \text{ } n2)$ We have the appropriate induction hypotheses for the terms $n1$ and $n2$ From the eq (1) we have

$$y = \text{apply}(n1 \text{ } n2) \quad \text{incr_free_var}(n1 \text{ } 1 \text{ } n1) \quad \text{incr_free_var}(n2 \text{ } 1 \text{ } n2) \quad (7)$$

From the eq (2)&(7)

$$w = \text{apply}(n1 \text{ } n2) \quad \text{subst}(x \text{ S}(1) n1 \text{ } n1) \quad \text{subst}(S(1) n2 \text{ } n2) \quad (8)$$

From the inductive hypotheses we have $n1 = n1$ and $n2 = n2$

Hence $w = \text{apply}(n1 \text{ } n2)$

(b) incr_incr lemma

The lemma shows how the order of two `incr_free_var` can be changed. The statement of this lemma is

$$\begin{aligned} & (i \ j \ \text{nat}) \ \text{le}(j \ i) \rightarrow \\ & \text{incr_free_var}(x \ i \ x1) \rightarrow \text{incr_free_var}(x1 \ j \ x1) \rightarrow \\ & \text{incr_free_var}(x \ j \ x2) \rightarrow \text{incr_free_var}(x2 \ S(i) \ x1 \ j) \end{aligned}$$

Proof The lemma is proved by induction on the structure of the term. We have

$$\text{le}(j \ i) \tag{1}$$

$$\text{incr_free_var}(i \ i) \tag{2}$$

$$\text{incr_free_var}(i \ j \ i) \tag{3}$$

$$\text{incr_free_var}(i \ j \ 2) \tag{4}$$

We have to show $\text{incr_free_var}(2 \ S(i) \ i)$

Variable case Different cases arise depending on the different values of i and j

For $i = v, j = v$: The variable v is free in eq (2). So $i = \text{variable}(S(v))$. This variable is again free in the eq (3). So $i = \text{variable}(v+2)$. The variable v in the eq (4) is free and so $x2 = \text{variable}(S(v))$. From the definition of `incr_free_var` we have that $\text{incr_free_var}(2 \ S(i) \ i)$ holds.

For $i = v, j \neq v$: This case is trivial.

For $i \neq v, j = v$: This case is trivial.

For $i \neq v, j \neq v$: In the eq (2) the variable v is bound and so $i = \text{variable}(v)$. This variable is free in the eq (3). So $i = \text{variable}(S(v))$. In the eq (4) the variable v is free and so $x2 = \text{variable}(S(v))$. From the definition of `incr_free_var` we have that $\text{incr_free_var}(x2 \ S(i) \ i)$ holds.

For $i=v, j=v$ _ In all the eq (2)-(4) the variables of substitution are bound So $1 = \text{variable}(v)$ $1 = \text{variable}(v)$ $2 = \text{variable}(v)$ Hence $\text{incr_free_var}(2 \ S(1) \ 1)$ holds

For $i=v, j \neq v$ _ This case is trivial

For $i \neq v, j=v$ _ The variable v in eq (2) is bound so $1 = \text{variable}(v)$ This variable in the eq (3) is free so $1 = \text{variable}(S(v))$ In the eq (4) the variable v is free so $2 = \text{variable}(S(v))$ From the definition of incr_free_var it holds that $\text{incr_free_var}(2 \ S(1) \ 1)$

For $i \neq v, j \neq v$ _ In all the eq (2)-(4) the variable is bound so we have $x1 = \text{variable}(v)$ $1 = \text{variable}(v)$ and $x2 = \text{variable}(v)$ Hence $\text{incr_free_var}(x2 \ S(1) \ 1)$

For $i \neq v, j \neq v$ _ This case is similar to the previous case

Lambda case Here we have $\text{lam } \lambda(m)$ From the inductive hypothesis for m the proof is straight forward

Apply case 1 The proof follows from the inductive hypotheses

•

(c) Incr_sbst_lemma1 :

This lemma shows how the incr_free_var distributes over the sbst Without the restriction $!t(i \ j)$ the lemma can not be proved

$$\begin{aligned} !t(i \ j) \rightarrow \text{sbst}(x \ j \ y \ w1) \rightarrow \text{incr_free_var}(w1 \ 2 \ w2) \rightarrow \\ \text{incr_free_var}(x \ x) \rightarrow \text{incr_free_var}(y \ 1 \ y) \rightarrow \\ \text{sbst}(x \ S(j) \ y \ w2) \end{aligned}$$

Proof : This lemma is proved by induction on the structure of the

tern y We have

$$lt(i, j) \quad (1)$$

$$bst(x, j, y, w1) \quad (2)$$

$$incr_free_var(w1, i, w2) \quad (3)$$

$$incr_free_var(i, j) \quad (4)$$

$$incr_free_var(y, i, y) \quad (5)$$

We have to prove that $sbst(x, S(j), y, w2)$

Variable case 1 We have $y = \text{variable}(v)$ The various cases for the different values of i, j are shown below

For $j = v, S(i) = v$ The variable v in eq (2) is free so $w1 = \text{Pred}(v)$ This variable in eq (3) is again free so $w2 = \text{variable}(v)$ The variable v in eq (5) is also free so $y = S(v)$ Hence we have $sbst(x, S(j), y, w2)$

For $j = v, S(i) \neq v$ This case is trivial

For $j \neq v, S(i) = v$ This case is also trivial

For $j \neq v, S(i) \neq v$ The variable v is the actual variable for substitution in eq (2) so $w1 = x$ So from the eq (3)&(4) $w2 =$ The variable in eq (5) is free so $y = \text{variable}(S(v))$ Hence $sbst(x, S(j), y, w2)$

For $j \neq v, S(i) = v$ The situation is similar to the previous case

For $j = v, S(i) \neq v$ This case is trivial

For $j \neq v, j \neq v$ The variable v is bound in eq (2) so $w1 = \text{variable}(v)$ This variable is free in eq (3) so $w2 = \text{variable}(S(v))$ The variable v is free in eq (5) so

$y = \text{variable}(S(v))$ Hence $\text{subst}(x \ S(j) \ y \ w2)$

For $j = v$ $j = v$ The variable v is bound in the eq (2) so
 $w1 = \text{variable}(v)$ This variable is again bound in eq (3) so
 $w2 = \text{variable}(v)$ The variable v is bound in eq (5) so
 $y = \text{variable}(v)$ Hence $\text{subst}(x \ S(j) \ y \ w2)$

For $j = v$, $j = v$ The situation is similar to the previous case

Lambda case 1 Here we have $y = \text{lambda}(\pi)$ From eq (2) there exists
 $x0 \ m0$ such that

$$w1 = \text{lambda}(m0) \quad \text{incr_free_var}(x0 \ 0 \ x0) \quad \text{subst}(x0 \ S(j) \ m \ m0) \quad (6)$$

From the eq (3)&(6) there exists $m1$ such that

$$w2 = \text{lambda}(m1) \text{ and } \text{incr_free_var}(\pi0 \ S(1) \ \pi1) \quad (7)$$

From the eq (5) there exists $m2$ such that

$$y = \text{lambda}(\pi2) \text{ and } \text{incr_free_var}(\pi \ S(i) \ \pi2) \quad (8)$$

We can find such that

$$\text{incr_free_var}(x0 \ 0 \ x0) \quad (9)$$

From incr_incr_lemma and eqs (4) (6)&(9) we conclude
 $\text{incr_free_var}(x0 \ S(1) \ x0)$ From the inductive hypothesis for m
 $\text{subst}(x \ S(S(j)) \ \pi2 \ \pi1)$ Hence from the eqs (7) (8)&(9) we can
conclude that $\text{subst}(x \ S(j) \ y \ w2)$

Apply case 1 We have $y = \text{apply}(\pi1 \ m2)$ From the eq (2)-(5) there
exists $\pi1 \ \pi1 \ m1 \ \pi2 \ \pi2 \ \pi2$ such that

$$w1 = \text{apply}(m1 \ \pi2) \quad \text{subst}(x \ j \ \pi1 \ m1) \quad \text{subst}(x \ j \ m2 \ \pi2) \quad (10)$$

$$w2 = \text{apply}(\pi1 \ m2) \quad \text{incr_free_var}(\pi1 \ 1 \ \pi1) \quad (11)$$

$$y = \text{apply}(\pi1 \ \pi2) \quad \text{incr_free_var}(\pi1 \ 1 \ m1) \quad (12)$$

From the inductive hypotheses for $\pi1$ and $m2$ we can infer
 $\text{subst}(x \ S(j) \ \pi1 \ \pi1)$ and $\text{subst}(y \ S(j) \ m2 \ m2)$ Hence from

the definition of sbst sbst(x S(j) y w2)

(d) Incr_sbst_lemma2 :

Th1 1 again another distribution of incr_free_var over sbst. The minor variation from incr_sbst_lemma1 has to be noticed carefully

$$\begin{aligned} le(1\ S(j)) \rightarrow sbst(m\ 1\ a\ w1) \rightarrow incr_free_var(w1\ j\ w2) \rightarrow \\ incr_free_var(a\ S(j)\ a\) \rightarrow incr_free_var(m\ j\ m) \rightarrow \\ sbst(m\ 1\ a\ w3) \rightarrow (w2=w3) \end{aligned}$$

Proof : The proof is by induction on the structure of the term a. We have $le(1\ S(j))$ (1)

sbst(m 1 a w1) (2)

incr_free_var(w1 j w2) (3)

incr_free_var(a S(j) a) (4)

incr_free_var(m j m) (5)

bst(m 1 a w3) (6)

Variable case _ We have a-variable(v) The various cases arising depending on various values i and j are shown below :

For $i \neq v$, $S(j) \neq v$ _ The variable v is free in the eq (2) so $w1 = \text{variable}(v-1)$ This variable is again free in eq (3) so $w2 = \text{variable}(v)$ The variable v in eq (4) is also free so $a = \text{variable}(v+1)$ This variable is again free in eq (6) so $w3 = \text{variable}(v)$ So $w2 = w3$

For $i \neq v$, $S(j) = v$ _ The variable v is free in the eq (2) so $w1 = \text{variable}(v-1)$ This variable is bound in eq (3) so $w2 = \text{variable}(v-1)$ The variable v is bound in eq (4) so $a = \text{variable}(v)$ This variable is free in eq (6) so $w3 = \text{variable}(v-1)$ So $w2 = w3$

For $i=v$, $S(j)=v$ _ The situation is very similar to the previous case

For $i=v$, $S(j)=v$ _ This case is trivial

For $i=v$, $S(j)=v$ _ The variable v in eq (2) is the actual variable to be replaced so $w1=\pi$ and $w2=\pi$ The variable v is bound in eq (4) so $a=variable(v)$ This variable in eq (6) is the actual variable for substitution so $w3=\pi$ Hence $w2=w3$

For $i=v$, $S(j)=v$ _ This case is similar to the previous one

For $i=v$, $S(j)=v$ _ This case is trivial

For $i=v$, $S(j)=v$ _ This case is also trivial

For $i=v$, $S(j)=v$ _ The variable v in eq (2) is bound so $w1=variable(v)$ The j can be greater than or equal to v So the variable in the eq (3) is bound so $w2=variable(v)$ The variable v is bound in eq (4) so $a=variable(v)$ This variable is again bound in eq (6) so $w3=variable(v)$ Hence $w2=w3$

Lambda case _ We have $a=lamba(m)$ From eq (2)-(6) there exist $n0 \ n1 \ m1 \ \pi2 \ \pi2$ such that

$$w1=lamba(\pi1) \ incr_free_var(\pi0 \ \pi0) \ sbst(\pi0 \ S(i) \ m \ \pi1) \quad (7)$$

$$w2=lamba(\pi1) \ incr_free_var(\pi1 \ S(j) \ \pi1) \quad (8)$$

$$a=lamba(\pi2) \ incr_free_var(\pi \ S(S(j)) \ \pi2) \quad (9)$$

$$w3=lamba(\pi2) \ incr_free_var(m \ 0 \ \pi) \ sbst(\pi \ S(i) \ \pi2 \ \pi2) \quad (10)$$

From $incr_incr_lemma$ and eq (5) (7) (10) we have

$$incr_free_var(m0 \ S(j) \ \pi) \quad (11)$$

From the inductive hypothesis for n we conclude that $m1 \ m2$. Hence $w2-w1$.

Apply case _ Here we have $a=apply(n1 \ n2)$. From the inductive hypotheses for $m1$ and $n2$ the proof of this case is straight forward.

(*) Lemma Incr_walk :

$$\begin{aligned} &incr_free_var(n1 \ i \ n1) \rightarrow incr_free_var(n2 \ i \ n2) \rightarrow \\ &walks(n1 \ n2) \rightarrow walks(n1 \ n2) \end{aligned}$$

This lemma tells us how the walks relation behaves with respect to $incr_free_var$. This lemma is needed in the proof of lambda case of $subst_lemma$.

Proof : The lemma is proved by induction on the structure of the term $n1$. We have

$$incr_free_var(n1 \ i \ n1) \tag{1}$$

$$incr_free_var(n2 \ i \ n2) \tag{2}$$

$$walks(n1 \ n2) \tag{3}$$

Variable case: Here we have $n1=variable(v)$. From eq (3)

$$n2=variable(v) \tag{4}$$

Three cases arise depending on the relation between i and v .

For $i=v$ _ The variable v is free in the eq (1)&(2). So $n1=variable(S(v))$ and $n2=variable(S(v))$. From the definition of $walks$ $n1$ walks to $n2$.

For $i \neq v$ _ The variable v is bound in the eq (1)&(2). So $n1=variable(v)$ and $n2=variable(v)$. From the definition of $walks$ $n1$ walks to $n2$.

For λ v _ the case is very similar to the previous one

Lambda case Here $n1 = \lambda a1 a2$ From the inductive hypothesis for term n and the definition of walk the proof is obvious

Apply case Here we have $n1 = \text{apply}(m1\ m2)$ From the eq (1) there exist $m1\ m2$ such that

$$n1 = \text{apply}(m1\ m2) \quad \text{incr_free_var}(m1\ i\ m1) \quad \text{incr_free_var}(m2\ i\ m2) \quad (5)$$

From the definition of walks two subcases arise:

Subcase (a) $n2 = \text{apply}(m3\ m4)$ walks($m1\ m3$) walks($m2\ m4$) (6)

From the eq (2)&(6) there exist $m3\ m4$ such that

$$n2 = \text{apply}(m3\ m4) \quad \text{incr_free_var}(m3\ i\ m3) \quad \text{incr_free_var}(m4\ i\ m4) \quad (7)$$

From inductive hypotheses for $m1\ m2$ and eq (6)&(7) walks($m1\ m3$) and walks($m2\ m4$) Hence $n1$ walks to $n2$

Subcase (b) There exist $a\ a1\ m3\ m4$ such that

$$m1 = \lambda a1 a2 \quad m3 = \lambda a1 a2 \quad \text{walks}(m1\ m3) \quad \text{walks}(m2\ m4) \quad \text{sbst}(m4\ \text{one}\ a1\ n2) \quad (8)$$

We can find terms $m3\ m4\ n2\ a1$ such that

$$\text{incr_free_var}(m3\ i\ m3) \quad (9)$$

$$\text{incr_free_var}(m4\ i\ m4) \quad (10)$$

$$m3 = \lambda a1 a2 \quad \text{incr_free_var}(a1\ 0(1)\ a1) \quad (11)$$

$$\text{sbst}(m4\ \text{one}\ a1\ n2) \quad (12)$$

From the lemma incr_sbst_lemma2

$$n2 = n2 \quad (13)$$

From the inductive hypothesis for $m1\ m2$ we infer that

$$\text{walks}(m1\ m3) \quad \text{walks}(m2\ m4) \quad (14)$$

From the definition of walk and eq (12)&(14) walks($n1\ n2$) Hence from eq (13) $n1$ walks to $n2$

(f) Sbst_sbst_lemma

This lemma shows how two successive bst operations can be exchanged. The restriction that $j \leq 1$ is warranted in the proof of the lemma.

$$\begin{aligned} &le(j, 1) \rightarrow sbst(m2, j, m1, w1) \rightarrow sbst(n2, 1, w1, w2) \rightarrow \\ &incr_free_var(n2, Pred(j), n2) \rightarrow sbst(n2, S(1), m1, m1) \rightarrow \\ &sbst(n2, 1, m2, m2) \rightarrow sbst(m2, j, m1, w3) \rightarrow (w2 = w3) \end{aligned}$$

When written in usual notation for $j \leq 1$

$$Ln2/i][[Ln2/j]m1 = L[Ln2/i]m2/j][[Ln2/S(1)]n1]$$

Proof : This lemma is proved by induction on the structure of the term $m1$.

$$\begin{aligned} &le(j, 1) & (1) \\ &sbst(m2, j, m1, w1) & (2) \\ &sbst(n2, 1, w1, w2) & (3) \\ &incr_free_var(n2, j, 1, n2) & (4) \\ &sbst(n2, (S(1)), m1, m1) & (5) \\ &sbst(n2, 1, m2, m2) & (6) \\ &sbst(m2, j, m1, w3) & (7) \end{aligned}$$

Variable case: For different values of $1, j$ the value of $w2$ and $w3$ are shown to be equal.

For $j < v, 1 < v, S(1) < v$: In the eq (2) $variable(v)$ is free so $w1 = variable(v-1)$. This variable is free in eq (3) so $w3 = variable(v-2)$. In eq (4) the $variable(v)$ is free so we have $m1 = variable(v-1)$. When $S(j) < v$ the $variable(v-1)$ in the eq (7) is free so $w3 = variable(v-2)$. So $w2 = w3$. The case $S(j) \geq v$ is not possible.

For $j < v, 1 < v, S(1) = v$: As in the previous case $w1 = variable(v-1)$

In the eq (3) $\text{variable}(v-1)$ is the variable for substitution So $w2=n2$ In eq (5) the $\text{variable}(v)$ is the variable for substitution so $m1=n2$ So from the lemma subst_not_free and eq (4)&(7) we have $w3=n2$ Hence $w2=w3$

For $j=v, i=v, S(1) v$ This case is trivial

For $j=v, i=v$ In the eq (2) the $\text{variable}(v)$ is free so $w1=\text{variable}(v-1)$ In the eq (3) this variable is bound so $w2=\text{variable}(v-1)$ In eq (5) the $\text{variable}(v)$ is bound so $m1=\text{variable}(v)$ This variable is free in eq (7) so $w3=\text{variable}(v-1)$ Hence $w2=w3$

For $j=v, i=v$ The situation is very similar to the previous case and we have $w2=w3 \text{ variable}(v-1)$

For $j=v, i=v$ This case is trivial

For $j=v, i=v$ In eq(2) the $\text{variable}(v)$ is the actual variable to be replaced so $w1=m2$ So $w2 = [n2/i]m2$ In the eq (5) the $\text{variable}(v)$ is bound so $m1=\text{variable}(v)$ This variable is the actual variable to be replaced in the eq (7) and so $w3 = m2$ i.e $w3 = [n2/i]m2$ Hence $w2=w3$

For $j=v, i=v$ The situation is very much same to the previous case We have $w2=w3=[n2/i]m2$

For $j=v, i=v$ This case is trivial

For $j=v, i=v$ This case is also trivial

For $j=v, i=v$ In all the eq (2) (3) (5)&(7) the variable v

is found Hence $w2-w3-variable(v)$

Larbia case Here we have $\pi1-lambda(m)$ The proof of this lemma
 e enplifies the utility of a proof checker in comple proving We
 have the induction lyptthesis that

$$\begin{aligned} & (m2 \pi1 \pi2 w1 w2 w3 n2 n2 term)(i j nat) \\ & le(j i) - sbst(\pi2 j m w1) - sbst(n2 i w1 w2) - \\ & incr_free_var(n2 j-1 n2) - sbst(n2 S(1) \pi \pi1) - \\ & sbst(n2 i m2 \pi2) - sbst(\pi2 j \pi1 w3) - w3 w2 \end{aligned} \quad (8)$$

From the definition of sbst and eq (2) there must exist
 $m21$ w11 such that

$$incr_free_var(n2 0 \pi21) \quad (9)$$

$$w1-lambda(w11) \quad (10)$$

$$sbst(m21 (j) \pi w11) \quad (11)$$

From the definition of sbst and eq (3)&(10) there exist $n21$ w21
 such that

$$incr_free_var(n2 0 n21) \quad (12)$$

$$w2-lambda(w21) \quad (13)$$

$$sbst(n21 S(1) w11 w21) \quad (14)$$

From the definition of sbst and eq (5) there exist $n2$ $\pi12$ such
 that

$$incr_free_var(n2 0 n2) \quad (15)$$

$$\pi1 = lambda(\pi12) \quad (16)$$

$$sbst(n2 S(S(1)) \pi \pi12) \quad (17)$$

From the definition of sbst and eq (7)&(16) there exist $m22$ w32
 such that

$$incr_free_var(\pi2 0 \pi22) \quad (18)$$

$$w3= lambda(w32) \quad (19)$$

$$sbst(\pi22 S(j) \pi12 w32) \quad (20)$$

From the incr_incr_lemma and eq (4) (12)&(15) we have that

$$incr_free_var(n21 j n2) \quad (21)$$

From the incr_sbst_lemma1 and eq (6) (9) (12)&(18) we have that

$$\text{subst}(n21 \ S(1) \ n21 \ m22) \quad (22)$$

From the induction hypothesis (8) and eq (11) (14) (17) (20) (21) (27)

$$w21=w32 \quad (23)$$

From the eq (13) (19)&(23) we have $w2=w3$

Apply case Here we have $n1=\text{apply}(m3 \ n4)$ We also have the appropriate induction hypotheses for the term $n3 \ n4$ From the definition of subst and eq (2) there exist $m3a \ m4a$ such that

$$w1=\text{apply}(n3a \ m4a) \quad (24)$$

$$\text{subst}(n2 \ j \ n3 \ m3a) \quad (25)$$

$$\text{subst}(n2 \ j \ n4 \ m4a) \quad (26)$$

From the definition of subst and eq (3)&(24) there exist $n3b \ m4b$ such that

$$w2=\text{apply}(n3b \ m4b) \quad (27)$$

$$\text{subst}(n2 \ i \ m3a \ m3b) \quad (28)$$

$$\text{subst}(n2 \ i \ n4a \ n4b) \quad (29)$$

From the definition of subst and eq (5) there exist $m3c \ n4c$ such that

$$n1=\text{apply}(n3c \ n4c) \quad (30)$$

$$\text{subst}(n2 \ S(1) \ n3 \ m3c) \quad (31)$$

$$\text{subst}(n2 \ S(1) \ n4 \ m4c) \quad (32)$$

From the definition of subst and eq (7) there exist $m3d \ n4d$ such that

$$w3=\text{apply}(m3d \ n4d) \quad (33)$$

$$\text{subst}(m2 \ j \ m3c \ m3d) \quad (34)$$

$$\text{subst}(n2 \ j \ n4c \ n4d) \quad (35)$$

From the induction hypothesis for $m3 \ n4$ and eq (24)-(35) we have that $n1b=n1d$ and $m2b=m2d$ Hence $w2=w3$

3 Subst_lemma : (Substitutivity of walks)

This lemma asserts that if $a1$ and $n1$ walk to $a2$ and $n2$

respectively then the result of substituting $n1$ in $a1$ for i walks to the result of substituting $n2$ in $a2$ for the same variable

$$\begin{aligned} \text{walks}(a1\ a2) \rightarrow \text{walks}(n1\ n2) \rightarrow \text{subst}(n1\ i\ a1\ w1) \rightarrow \\ \text{subst}(n2\ i\ a2\ w2) \rightarrow \text{walks}(w1\ w2) \end{aligned}$$

This lemma is crucial for the proof that the walks relation follows diamond property

Proof

This lemma is proved by induction on the structure of the term $a1$. Here we have

$$\text{walks}(a1\ a2) \tag{1}$$

$$\text{walks}(n1\ n2) \tag{2}$$

$$\text{subst}(n1\ i\ a1\ w1) \tag{3}$$

$$\text{subst}(n2\ i\ a2\ w2) \tag{4}$$

Variable case Here we have $a1 = \text{variable}(v)$. From the definition of walks and eq (1) we have $a2 = \text{variable}(v)$. Here three cases arise depending on the relation between i and v .

For $i \neq v$: In the eq (3)&(4) the variable v is free so from the definition of subst we have $w1 = \text{Pred}(v)$ $w2 = \text{Pred}(v)$. So from the definition of walks $w1$ walks to $w2$.

For $i = v$: In both the eq (3) and (4) the variable v is the actual variable to be substituted. So from the definition of walks $w1 = n1$ and $w2 = n2$. From eq (2) $w1$ walks to $w2$.

For $i \neq v$: In both the eq (3) and (4) the variable v is bound. So from the definition of subst $w1 = \text{variable}(v)$ and $w2 = \text{variable}(v)$. Hence $w1$ walks to $w2$.

Lambda case Here we have $a1 = \text{lambda}(m)$. From the definition of walks

and eq (1) there e 1st m1 such that

$$a2 = \text{landa}(\pi 1) \text{ and } \text{walks}(\pi \pi 1) \quad (5)$$

From the definition of sbst and eq (3) (4)&(5)

$$\text{incr_free_var}(n1 \ 0 \ n1) \ w1 = \text{landa}(w1) \ \text{sbst}(n1 \ S(1) \ \pi \ w1) \quad (6)$$

$$\text{incr_free_var}(n2 \ 0 \ n2) \ w2 = \text{landa}(w2) \ \text{sbst}(n2 \ S(1) \ m \ w2) \quad (7)$$

From the lemma incr walk and eq (6)&(7)

$$\text{walks}(n1 \ n2) \quad (8)$$

From the inductive hypothesis for m and eq (5) (7) we infer walks(w1 w2) Hence from eqs (6)&(7) walks(w1 w2)

Apply case! Here we have a1=apply(m1 $\pi 2$)

Here two cases arise from the definition of walk

Subcase (a)! There e 1st $\pi 3$ and m4 such that

$$a2 = \text{apply}(\pi 3 \ \pi 4) \ \text{walks}(\pi 1 \ \pi 3) \ \text{walks}(m2 \ m4) \quad (9)$$

From the definition of sbst eq (3)&(4) there e 1st $\pi 1 \ \pi 2 \ \pi 3 \ \pi 4$ such that

$$w1 = \text{apply}(\pi 1 \ \pi 2) \ \text{sbst}(n1 \ i \ \pi 1 \ \pi 1) \ \text{bst}(n2 \ i \ \pi 2 \ \pi 2) \quad (10)$$

$$w2 = \text{apply}(m3 \ \pi 4) \ \text{sbst}(n1 \ i \ m3 \ \pi 3) \ \text{sbst}(n2 \ i \ \pi 4 \ \pi 4) \quad (11)$$

From the inductive hypotheses for m1 and $\pi 2$ we infer that m1 walks to m3 and $\pi 2$ walks to $\pi 4$ Now from the definition of walks w1 walks to w2

Subcase (b) There e 1st terms a a1 $\pi 3 \ m4$ such that

$$\begin{aligned} \pi 1 &= \text{landa}(a) \ \pi 3 = \text{landa}(a1) \ \text{walks}(a \ a1) \ \text{walks}(\pi 1 \ \pi 3) \\ \text{walk}(\pi 2 \ \pi 4) &\ \text{sbst}(\pi 4 \ \text{one} \ a1 \ a2) \end{aligned} \quad (12)$$

From the eq (3) and definition of sbst there e 1st $\pi 1 \ \pi 2$ such that

$$w1 = \text{apply}(\pi 1 \ \pi 2) \ \text{sbst}(n1 \ i \ \pi 1 \ \pi 1) \ \text{sbst}(n2 \ i \ \pi 2 \ m2) \quad (13)$$

We can find terms n2 $\pi 4 \ a1 \ w$ such that

$$\text{incr_free_var}(n2 \ 0 \ n2) \quad (14)$$

$$\text{sbst}(n2 \ i \ \pi 4 \ \pi 4) \quad (15)$$

sbst(n2 S(1) a1 a1) (16)

sbst(m4 one a1 w) (17)

From the eq (13) (17) and lemma sb t sb t_lemma we can infer that $w = w2$. From the inductive hypotheses for $m1 = m2$ we have

$walks(m1 \text{ lamda}(a1))$ and $walks(m2 \text{ n4})$ (18)

From the definition of walks and eq (17)&(18)

$walks(apply(m1 \text{ m2}) w)$ (19)

Since $w = w2$ from the eq (19) we have $walks(w1 \text{ w2})$

4 Walks follows diamond property (Theorem diamond_walks)

In this section we prove that the relation walks follows diamond property. The theorem diamond_walks reads as

(x term) diamond (x)

It states that all lambda calculus terms can be the sources of diamonds. Another form of the statement is

$walks(x \text{ y})$ and $walks(x \text{ z}) \Rightarrow$
 $\exists (w \text{ term})(walks(y \text{ w}) \wedge walks(z \text{ w}))$

Proof: The proof is by induction on the structure of the term x i.e. the term which is being walked from. We make use of `sub_term_induction` instead of `term_induction`. The `sub_term_induction` is a stronger version of induction on the structure of the term. The use of `sub_term_induction` is actually necessitated in the case when $= apply(m \text{ n})$ $m = \text{lamda}(m')$ for some $n = n'$ and the last redex β -reduced in the walk is $apply(m' \text{ n})$. In this case if we had made use of `term_induction` we have the induction hypothesis only for the term m . But we need a stronger induction hypothesis holding for the sub_term m' of m . It is sure that we can prove that the term m' can become the source of a diamond if we have that m' is the source of a diamond and $m = \text{lamda}(m')$ once we have `walk_diamond`. But we are actually proving the diamond property itself and hence can not deduce that m' can

be the source of a diamond if $n = 1$ one suff. So we need the stronger version of induction

We have $walks(x, y)$ and $walks(x, z)$ (1)

Variable case : We have : $x = variable(v)$. If x walk to y and from the definition of $walks$ we have that

$$y = variable(v) \text{ and } z = variable(v) \quad (2)$$

Choose $w = variable(v)$. From the definition of $walks$ and from the eq(2) we have that both y and z walk to w

Lambda case : We have : $x = lambda(n)$. If x walks to y and from the definition of $walks$ there must exist terms $m1$ and $m2$ such that

$$y = lambda(m1) \text{ and } walks(m, m1) \quad (3)$$

$$z = lambda(m2) \text{ and } walks(m, m2) \quad (4)$$

Induction hypothesis says that all the sub_terms of m and n can be the sources of diamonds. This means that m and n themselves can be the sources of the diamonds. From this and eq (3) and eq (4) it follows that there must exist a term $m3$ such that

$$walks(m1, m3) \text{ and } walks(m2, m3) \quad (5)$$

Choose $w = lambda(m3)$. From the definition of $walks$ and eq (3) (4)&(5) we have that both y and z walk to w

Apply case : Here we have : $x = apply(m, n)$. We have $walks(x, y)$ and $walks(x, z)$. From the definition of $walks$ there arise four subcases depending on whether or not m is an abstraction and if it is whether or not the redex $apply(m, n)$ is the last redex to be β -reduced in the walk

Subcase (a) $Apply(m, n)$ is not the last redex to be β -reduced in both the walks to y and z . From the definition of $walks$ if x walks to y and z then there must exist terms $m1, n1, m2, n2$ such that

$$y = \text{apply}(m1\ n1) \text{ and } = \text{apply}(m2\ n2) \quad (6)$$

$$\text{walks}(m\ m1) \text{ and } \text{walks}(n\ n1) \quad (7)$$

$$\text{walks}(m\ m2) \text{ and } \text{walks}(n\ n2) \quad (8)$$

We have the induction hypotheses that all subterms of m and n can be the sources of diamonds. This obviously means that the terms m and n can themselves be the sources of diamonds. From this conclusion and eq (7)&(8) there must exist $w1$ and $w2$ such that

$$\text{walks}(m1\ w1) \text{ and } \text{walks}(m2\ w1) \quad (9)$$

$$\text{walks}(n1\ w2) \text{ and } \text{walks}(n2\ w2) \quad (10)$$

Now choose $w = \text{apply}(w1\ w2)$. From eq (6) (9)&(10) and from the definition of walks we can conclude that y of the form $\text{apply}(m1\ n1)$ and of the form $\text{apply}(n2\ n2)$ walks to w of the form $\text{apply}(w1\ w2)$.

Subcase (b) $\text{Apply}(m\ n)$ is the last redex to be β -reduced in the walk to x while it is not in the walk to y . From the definition of walks if walks to y then there must exist terms $m1$ and $n1$ such that

$$y = \text{apply}(m1\ n1) \quad \text{walks}(m\ m1) \text{ and } \text{walks}(n\ n1) \quad (11)$$

From the definition of walks and the eq (11) there must exist a term $a1$ such that

$$m1 = \text{lambda}(a1) \text{ and } \text{walks}(a\ a1) \quad (12)$$

From the definition of walk if walks to x then there must exist terms $a\ a2\ m2\ n2$ such that

$$\begin{aligned} m &= \text{lambda}(a) \quad m2 = \text{lambda}(a2) \quad \text{walks}(a\ a2) \quad \text{walks}(m\ m2) \\ &\quad \text{walks}(n\ n2) \text{ and } \text{subst}(n2\ \text{one } a2) \end{aligned} \quad (13)$$

From the induction hypotheses that all the sub-terms of m and n can be the sources of the diamonds the terms $a\ n$ can be the sources of diamonds. From this and the eq (11) (12)&(13) there exist terms $a3\ n3$ such that

$$\text{walks}(a1\ a3) \text{ and } \text{walks}(a2\ a3) \quad (14)$$

$$\text{walks}(n1\ n3) \text{ and } \text{walks}(n2\ n3) \quad (15)$$

Now choose a term w such that $\text{subst}(n3 \text{ one } a3 \ w)$. From the definition of walks and the eq (11) (12) (14)&(15) we conclude that the term y of the form $\text{apply}(n1 \ n1)$ walks to the term w of the form Ln3/1Ja3 . From the subst_lemma (substitutivity of walks) and eq (13) (14) (15) we conclude that the term of the form Ln2/1Ja walk to the term w of the form Ln3/1Ja3 .

Subcase (c) $\text{Apply}(n \ n)$ is the last redex to be β -reduced in the walk to y while it is not in the walk to . The proof is similar to the last case

Subcase (d) $\text{Apply}(n \ n)$ is the last redex to be β -reduced in both the walks to y and . From the definition of walks there must exist terms $a \ a1 \ m1 \ n1 \ a0 \ a2 \ m2 \ n2$ such that

$$\begin{aligned} m &= \text{lamda}(a) \quad m1 = \text{lamda}(a1) \quad \text{walks}(a \ a1) \\ \text{walks}(n \ m1) & \quad \text{walks}(n \ n1) \quad \text{subst}(n1 \text{ one } a1 \ y) \end{aligned} \quad (16)$$

$$\begin{aligned} n &= \text{lamda}(a0) \quad n2 = \text{lamda}(a2) \quad \text{walks}(a0 \ a2) \\ \text{walks}(m \ m2) & \quad \text{walks}(n \ n2) \text{ and } \text{subst}(n2 \text{ one } a2 \) \end{aligned} \quad (17)$$

Since $n = \text{lamda}(a) = \text{lamda}(a0)$ we have $a = a0$. Hence $\text{walks}(a \ a2)$ (18)

The induction hypotheses say that all the sub-terms of the terms m and n can be the sources of diamonds. The terms a and n can be sources of diamonds. From this and eq (16) eq (17) eq (18) we can conclude that there exist terms $a3 \ m3$ such that

$$\text{walks}(a1 \ a3) \text{ and } \text{walks}(a2 \ a3) \quad (19)$$

$$\text{walks}(n1 \ n3) \text{ and } \text{walks}(n2 \ n3) \quad (20)$$

Now choose a term w such that $\text{subst}(n3 \text{ one } a3 \ w)$. From the subst_lemma (substitutivity of walks) eq (16) eq (17) eq (18) eq (19) eq (20) we can infer that y of the form Ln1/1Ja1 and the form Ln2/1Ja2 walks to w of the form Ln3/1Ja3 .

5 Reflexive Transitive Closure of Walks & Diamond Property :

The relation n_walks is the reflexive and transitive closure

of walks relation. The main result of the section is that n_walks also follows diamond property. Before proving this we prove a small lemma.

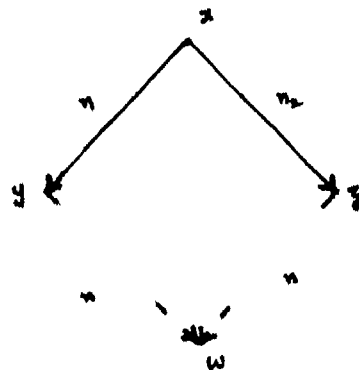
5.1 Lemma $m_one_diamond_one_m$:

$$n_walks(x\ y\ n) \rightarrow walks(x\ z) \rightarrow \\ \exists (w\ term) (walks(y\ w) \wedge n_walks(z\ w\ n))$$

This lemma is a special case of the subsequent theorem in the sense that the length of second walk chain is fixed at one. Hence the proof is not given.

5.2 Theorem $trans_diamond$:

$$n_walks(x\ y\ n1) \rightarrow n_walks(x\ z\ n2) \rightarrow \\ \exists (w\ term) n_walks(y\ w\ n2) \wedge n_walks(z\ w\ n1)$$



The statement of this theorem can be shown by the diagram above.

Proof : The theorem is proved by a double induction on the lengths of walk chains from x to y and z in $n1$ and $n2$. We have

$$n_walks(x\ y\ n1) \quad \text{and} \quad n_walks(x\ z\ n2) \quad (1)$$

Case 1 (ero_case) The lengths of walk chains are ero. If x walks to y and z in 0 steps from the definition of n_walks for reflexive case

$$y = \text{and} = \quad (2)$$

Now choose $w =$ Now from the definition of n_walks and eq (2) we know that $n_walks(y \ w \ 0)$ and $n_walks(\ \ w \ 0)$

Case 2 (ind1_case) We have the induction hypothesis that for one fixed $n1 \ n2$

$$\begin{aligned} & (\ \ y \ \text{term}) \ n_walks(\ \ y \ n1) = n_walks(\ \ n2) = \\ & \exists(w \text{ term}) (\ n_walks(y \ w \ n2) \wedge n_walks(\ \ w \ n1)) \end{aligned} \quad (3)$$

We are given that for a particular \langle

$$n_walks(\langle \ y \ S(n1) \rangle) \text{ and } n_walk(\langle \ z \ n2 \rangle) \quad (4)$$

Now we have to exhibit a w such that y and w walk to it in $n2$ and $S(n1)$ walk steps respectively From eq (4) and the definition of n_walks for transitive case there must exist a term i such that

$$n_walks(\ \ \langle i \ n \rangle) \text{ and } walks(\langle i \ y \rangle) \quad (5)$$

From the induction hypothesis and eq (5) there exists a term $w1$ such that

$$n_walks(\langle i \ w1 \ n2 \rangle) \text{ and } n_walks(\ \ w1 \ n1) \quad (6)$$

From the eq (5)&(6) and $m_one_diamond_one_n$ there exists a term $w2$ such that

$$n_walks(y \ w2 \ n2) \text{ and } walks(w1 \ w2) \quad (7)$$

Choose $w=w2$ From the eq (6)&(7) and definition of n_walk for transitive case we know that $n_walks(y \ w \ n2)$ and $n_walks(\ \ w \ S(n1))$

Case 3: (ind2_case) The proof is similar to the previous case except that y and $n1$ and $n2$ are interchanged

Case 4 (ind3_case)

We have the induction hypothesis that for fixed $n1$ and $n2$

$$\begin{aligned} & (\langle \ y \ \text{term} \rangle \ n_walks(\langle \ y \ y \ n1 \rangle) = n_walks(\ \ n2) = \\ & \exists(w \text{ term}) (\ n_walks(y \ w \ n2) \wedge n_walks(\ \ w \ n1)) \end{aligned} \quad (8)$$

We are given that for fixed y

$$n_walks(x \ y \ S(n1)) \text{ and } n_walks(\quad S(n2)) \quad (9)$$

We have to exhibit a w such that y and w walk to z in $S(n2)$ and $S(n1)$ walk steps. From the definition of n_walks for transitive case and eq (9) there exist terms $x_1 \ x_2$ such that

$$n_walks(x_1 \ n1) \text{ and } walks(x_1 \ y) \quad (10)$$

$$n_walks(x_2 \ n2) \text{ and } walks(x_2 \ z) \quad (11)$$

From the induction hypothesis and eq (10)&(11) there exist w_1 such that

$$n_walks(x_1 \ w_1 \ n2) \text{ and } n_walks(x_2 \ w_1 \ n1) \quad (12)$$

From the lemma_one_diamond_one_n and eq (10)&(11)&(12) there exist w_2 and w_3 such that

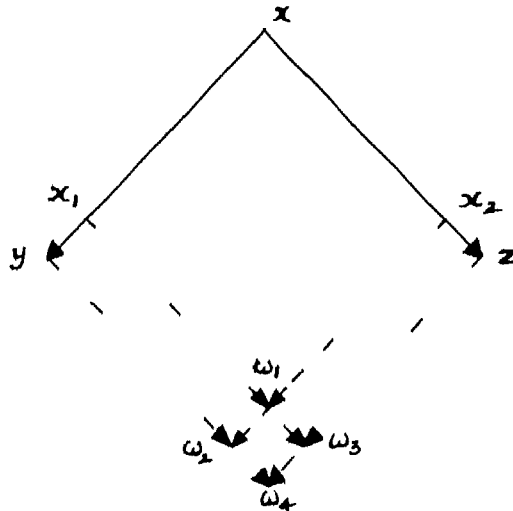
$$n_walks(y \ w_2 \ n2) \text{ and } walks(w_1 \ w_2) \quad (13)$$

$$n_walks(w_3 \ n1) \text{ and } walks(w_1 \ w_3) \quad (14)$$

From theorem diamond_walk and eq (13)&(14) there exists a term w_4 such that

$$walks(w_2 \ w_4) \text{ and } walks(w_3 \ w_4) \quad (15)$$

Choose $w = w_4$. Hence from the definition of n_walks for transitive case and eq (13) (14)&(15) we conclude that $n_walks(y \ w \ S(n2))$ and $n_walks(w \ S(n1))$



The whole construction of w can be shown in the above diagram

6 Church-Rosser

The key of the proof lies in showing that a b-step can be represented as a walk and a walk can be represented as a series of b steps. This actually amounts to saying that b-reduction can be seen as reflexive transitive closure of walks relation. We have already seen how a particular reflexive transitive closure relation of walks relation the n_walks relation follows diamond property.

6.1 Lemma b_step_to_walks :

$$b_step(u\ v) \rightarrow walks(u\ v)$$

Proof : From the definition of b_step u can be an abstraction or an application. For a particular u and v we have

$$b_step(u\ v) \tag{1}$$

Lambda case: Here we have $u = \lambda x(\pi)$. From the definition of b_step there exists $m1$ such that $v = \lambda x(m1)$ and $b_step(\pi\ m1)$. From the definition of walks and inductive hypothesis for π we have $walks(u\ v)$.

Apply case Here we have $u = apply(n1\ m2)$ and appropriate inductive hypotheses for terms $n1$ and $m2$. From the definition of b_step three cases arise:

Subcase (a): Here $n1$ b_steps to $n1$. So we have

$$v = apply(n1\ m2) \text{ and } b_steps(m1\ n1) \tag{2}$$

From the inductive hypothesis and eq (2) we have

$$walks(\pi1\ n1) \text{ and } walks(m2\ m2) \tag{3}$$

Hence from the definition of walks we have $walks(u\ v)$.

Subcase (b) : Here $m2$ b_steps to $n2$. This case is similar to the previous one.

Subcase (c) — Here we have $m1 = \text{lamda}(a)$ and $\text{bst}(m2 \text{ one } a \ v)$. From the definition of walk we have $\text{walks}(u \ v)$

6 2 Lemma $n_b_steps_to_n_walks$:

$$n_b_step(u \ v \ n) \rightarrow n_walks(u \ v \ n)$$

We know from the previous lemma that a b step can be represented as a walk step. Current lemma says that a series of b steps can be seen as an equal number of walk steps. Current lemma can be proved by the induction on the length of reduction chain and from the fact that n_walks and n_b_step are reflexive transitive closures of walks and b_step respectively.

6 3 Lemma $abs_same_b_steps$:

$$n_b_step(u \ v \ n) \rightarrow n_b_step(\text{lamda}(u) \ \text{lamda}(v) \ n)$$

This lemma follows straight forwardly from the definition of n_b_step and induction on the length of reduction.

6 4 Lemma $b_step_trans_add$:

$$n_b_step(u \ v \ n1) \rightarrow n_b_step(v \ w \ n2) \rightarrow n_step(u \ w \ \text{add}(n1 \ n2))$$

This lemma can be proved by induction on either $n1$ or $n2$.

6 5 Lemma $app_add_b_steps$:

$$\begin{aligned} n_b_step(m1 \ m3 \ n1) &\rightarrow n_b_step(m2 \ m4 \ n2) \rightarrow \\ n_b_step(\text{apply}(m1 \ m2) \ \text{apply}(m3 \ m4) \ \text{add}(n1 \ n2)) \end{aligned}$$

Proof : This lemma is proved by double induction on $n1 \ n2$. We have $n_b_step(m1 \ m3 \ n1)$ and $n_b_step(m2 \ m4 \ n2)$ (1)

Case 1: (Base case) When $n1=0$ $n2=0$ we have $m1=m3$ and $m2=m4$ from the definition of n_b_step . So $\text{apply}(m1 \ m2)$ reduces in zero steps to $\text{apply}(m3 \ m4)$.

Case 2 (Induction step for n1) We have

$$n_b_step(m1\ m3\ S(n1)) \text{ and } n_b_step(m2\ m4\ n2) \quad (2)$$

We also have the inductive hypothesis for n1 and n2. We have to prove that

$n_b_step(apply(m1\ m2)\ apply(m3\ m4)\ add(S(n1)\ n2))$ holds. From the definition of n_walks and eq (2) there exists $m1$ such that

$$n_b_step(m1\ m1\ n1) \text{ and } b_step(m1\ m3) \quad (3)$$

From the inductive hypothesis and eq (2)&(3) we know that

$$n_b_step(apply(m1\ m2)\ apply(m1\ m4)\ add(n1\ n2)) \quad (4)$$

From the definition of b_step and eq (3) we have

$$b_step(apply(m1\ m4)\ apply(m3\ m4)) \quad (5)$$

From the definition of n_b_step and eq (4)&(5) we have

$$n_b_step(apply(m1\ m2)\ apply(m3\ m4)\ S(add(n1\ n2))) \quad (6)$$

From this it is obvious that

$$n_b_step(apply(n1\ m2)\ apply(m3\ m4)\ add(S(n1)\ n2))$$

Case 3&4 — These two cases are generalisations of the last case

6.6 Lemma walks_to_b_steps :

$$walks(u\ v) \rightarrow \exists (n\ nat) (n_b_step(u\ v\ n))$$

This lemma says that a walk step can be represented as a series of b-steps. We have

$$walks(u\ v) \quad (1)$$

Proof : The proof is by induction on the structure of the term u.

Variable case: Here we have $u = variable(v0)$. From the definition of walk

$$v = variable(v0) \quad (2)$$

From the definition of $refl_b_step$ we conclude that the required n is zero.

Lambda case We have $u = \text{lamda}(\pi)$ The inductive hypothesis is

$$(v : \text{term}) \text{ walks}(\pi \ v) \rightarrow \exists (n : \text{nat}) \ n_b_step(\pi \ v \ n) \quad (3)$$

From the definition of walk there exists $n1$ such that

$$\text{walks}(\pi \ n1) \text{ and } v = \text{lamda}(n1) \quad (4)$$

From eq (3)&(4) there exists n such that $n_b_step(\pi \ n1 \ n)$ from
 $\text{abs_same_b_steps} \ n_b_step(\text{lamda}(m) \ v \ n)$

Apply case We have $u = \text{apply}(m1 \ m2)$ We have to exhibit an n such
that $n_b_step(\text{apply}(m1 \ m2) \ v \ n)$ We have the inductive hypotheses
for $m1$ and $m2$ that

$$(v : \text{term}) \text{ walk} \ (\pi1 \ v) \rightarrow \exists (n : \text{nat}) \ n_b_step(\pi1 \ v \ n) \quad (5)$$

$$(v : \text{term}) \text{ walks}(\pi2 \ v) \rightarrow \exists (n : \text{nat}) \ n_b_step(\pi2 \ v \ n) \quad (6)$$

From the definition of walk two cases arise:

Subcase(a) There exist $m3 \ m4$ such that

$$\text{walks}(m1 \ m3) \text{ and } \text{walks}(m2 \ m4) \quad (7)$$

From the inductive hypotheses there exist $n1 \ n2$ such that

$$n_b_step(\pi1 \ m3 \ n1) \text{ and } n_b_step(\pi2 \ m4 \ n2) \quad (8)$$

From app_add_b_steps $\text{add}(n1 \ n2)$ is the required n

Subcase(b) There exists a $a1 \ m3 \ m4$ such that

$$m1 = \text{lamda}(a) \quad m3 = \text{lamda}(a1) \quad \text{walks}(\pi1 \ m3) \\ \text{walks}(m2 \ m4) \quad \text{subst}(\pi4 \ \text{one } a1 \ v) \quad (9)$$

From the inductive hypotheses there exist $n1 \ n2$ such that

$$n_b_step(m1 \ m3 \ n1) \text{ and } n_b_step(m2 \ m4 \ n2) \quad (10)$$

From lemma app_add_b_steps and eq (10)

$$n_b_step(\text{apply}(m1 \ m2) \ \text{apply}(m3 \ m4) \ \text{add}(n1 \ n2)) \quad (11)$$

From the definition of b_step and eq (9)

$$b_step(\text{apply}(\pi3 \ m4) \ v) \quad (12)$$

From the definition of n_b_step and eq (11) (12) we infer
 $n_b_step(\text{apply}(m1 \ m2) \ v \ S(\text{add}(n1 \ n2)))$ and so $S(\text{add}(n1 \ n2))$ is the
required n

6.7 Lemma $n_walks_to_n_b_steps$:

$$n_walks(u\ v\ n) \rightarrow \exists (m\ nat) n_b_step(u\ v\ m)$$

The previous lemma showed how a single walk step can be seen as a number of b steps. This lemma says that a walk chain can be seen as a series of b steps. The proof is by induction on the length of the walk chain.

6.8 Theorem Church-Rosser

For all terms $x\ y$ and z $reduces(x\ y) \rightarrow reduces(x\ z) \rightarrow$

$$\exists (w\ term) reduces(y\ w) \wedge reduces(z\ w)$$

Proof : After unfolding the definition of `reduces` there exist $n1\ n2$ such that

$$n_b_steps(x\ y\ n1) \tag{1}$$

$$n_b_steps(x\ z\ n2) \tag{2}$$

We have to exhibit a w such that y and z reduce to it. From lemma `n_b_steps_to_n_walks` we conclude that

$$n_walks(x\ y\ n1) \tag{3}$$

$$n_walks(x\ z\ n2) \tag{4}$$

From the lemma `trans_diamond` and eq (3)&(4) there must exist a w such that $n_walks(y\ w\ n2)$ and $n_walks(z\ w\ n1)$ (5)

From the lemma `n_walks_to_n_b_steps` there must exist $m1\ m2$ such that $n_b_step(y\ w\ m1)$ and $n_b_step(z\ w\ m2)$. Hence proved.

CHAPTER 4

CONCLUDING REMARKS

In this chapter we briefly mention the arguments for proof checking limitations of the present exercise problems encountered with the system and the future work possible in this direction

The de Bruijn notation is more suitable for computer implementations of λ -calculus. With this notation we do away with α -reduction. The present study stresses the fact that no essential part of λ calculus is lost by switching to this notation.

The proof is presented in a goal directed manner. Goal directed proving is the reverse of natural deduction proving. In the natural deduction we start with the axioms and reduce them to the goal. In the goal directed proving we start with the goal and try to reduce the goal into smaller and smaller subgoals until all the subgoals are reduced to axioms.

In the present study we started with the main theorem of Church Rosser as the goal. As we tried to reduce this goal into smaller and smaller subgoals we came to the realisation that we needed some more lemmas to proceed further. So these lemmas were taken as axioms at that point of time. These axioms were taken up later which in turn demanded some more lemmas. For example consider the proof of the lemma that a single walk allows the diamond property. This needs the `subst_lemma` (substitutivity of walks) in the apply case. The proof of the `subst_lemma` in turn demands a lemma `subst_subst_lemma` in the apply case. This in turn demands the lemma `incr_subst_lemma1` which in turn demands `incr_incr_lemma`. In the machine proof these lemmas are developed in the above mentioned order i.e. `subst_lemma` `subst_subst_lemma` `incr_subst_lemma1` and `incr_incr_lemma`. But in the proof presented

in the chapter these lemmas are presented in the reverse order i.e. in natural deduction style. So the goal directed proof presents a goal directed approach since we do not know at the beginning what lemmas are needed for the proof. We start with the goal and prove the lemmas as and when needed.

One more advantage of the proof checking by machine is clear when the proof becomes very big, complex and unwieldy for the humans. The proof of `subst_bst_lemma` for lambda case clearly shows the advantage of having a proof checker at hand.

For the proof of the Church-Rosser, not much higher types are needed. The proof could have gone through in a first order system also. The proof of strong normalisation would have shown more clearly the necessity of an higher order system.

One problem with the CUC V 4.10 is the parsing stack overflow. The tactics theorem prover does not do the complete type checking. Only after the complete proof is gone through when we try to save the theorem the complete type checking is done. At this point many of the large proofs (involving 20 to 30 existential eliminations) resulted in parsing stack overflows. So the bigger proofs have to be broken into small unnatural segments.

We have encountered another difficulty with the induction definition facility. We wanted a recursive definition of `subst` operation of type `term -> nat -> term -> term`. This can not be done using the inductive definition facility. Inductive definition allows the definition of an altogether new types. But `subst` is an element, not a type. In order to define the `subst` we have to define it by the predicate `subst : term -> nat -> term -> term -> Prop` which is the characteristic function of the substitution.

Future work possible in this direction is to code the proofs of strong normalisation of typed λ -calculus, standardisation theorem and undecidability of the convertibility for pure λ -calculus.

REFERENCES

- [BM77] Boye L, Morla J. A computational logic. Academic press. Milano. It. 1979
- [Ch40] Church A. A formulation of simple theory of types. Journal of symbolic Logic 5 1940
- [CH88] Coquand T, Huet G. The calculus of constructions. Information and computation 76 pp 95-120 1988
- [Co86] Constable R L et al. Implementing mathematics with NUIRI. proof development system. Prentice Hall Inc. Englewood Cliffs. New Jersey 1986
- [Cq86] Coquand T. An analysis of Girard paradox. LICS Boston 1986
- [Cq87] Coquand T. Metamathematical investigations of a calculus of constructions. Documentation and User's guide. Rapports Techniques 110 1989 INRIA France
- [Cq89] Coquand T. The tactics theorem prover. Documentation and User's Guide. Rapports Techniques No 110 1989 INRIA
- [Sh88] Shankar N. A mechanical proof of the Church-Rosser theorem. JACM vol 35 no 3 July 1988 pp 475-522
- [Ba80] Barendregt H P. The lambda calculus. North Holland Amsterdam 1980
- [dB72] de Bruijn N G. Lambda calculus notation with nameless dummies: a tool for automatic formula manipulation with application to Church-Rosser theorem. Indag Math 34 5(1972) pp 381-392
- [dB80] de Bruijn N G. A survey of the Projet AUTOMATH. In essays on Combinatory logic. Lambda calculus and Formalism. J P Seldin and J R Hindley Eds. Academic press Orlando Fla 1980 pp 589-606
- [D89] Dowek G. A vernacular Syllabus. Documentation and User's Guide. Rapports Techniques No 110 1989 INRIA

- combinatory logic Cambridge University press London 1972
- [BH70] Barendregt H Henerick K Types in Lambda calculi and programming languages Technical report no 90-4 Feb 1990 University of Nijmegen
- [H87] Huet G A uniform approach to type theory In Logical foundations of functional programming University of Texas June 1987
- [H89] Huet G Constructive Engine Documentation and User's Guide Rapports Techniques No 110 1989 INRIA
- [Ha87] Harper R F et al A framework for defining logics Proceedings of second symp LICS 1984 pp 194-204
- [PM89] Paulin-Mohring Inductive definitions in the Calculus of Constructions Documentation and User's Guide Rapports Technique No 110 1989 INRIA

APPENDIX A HOW TO USE COC

When we execute `constr` command we enter into CAML interactive mode. This is indicated by the prompt `#`. In this mode we can verify the proofs contained in `constr` files by the command

`V` name of the file

We can enter the Vernacular mode from the CAML mode by typing `goC()`. The prompt for Vernacular mode is `->`. All the commands of the CAML interactive mode end with double semicolon. All the commands of the Vernacular interactive mode end with the full-stop.

A typical session of the system in Vernacular mode is shown below. The bold text is typed by the user and italic text is the system response. We define inductively the type *nat* of natural numbers and the relation *le* of type `nat -> nat -> Prop`. Then we prove the lemma that if two numbers are in *le* relation their successors are also in *le*. This lemma is needed in the machine proof for proving the properties of de Bruijn indices.

We define *nat* inductively as

- Inductive Data nat = 0 : nat | S : nat -> nat

nat is defined

0 is defined

S is defined

nat_REC is defined

nat_rec is defined

We define relation *le* inductively as

-> Inductive Definition le [n:nat] : nat -> Prop

-> = le_n : (le n n)

```

-> | le_S : (m:nat)(le n m) -> (le n (S m))
le is defined
le_n is defined
le_S is defined
le_rec is defined
le : nat -> nat -> Prop
le_n : (n:nat)(le n n)
le_S : (n:nat)(m:nat)(le n m) -> (le n (S m))
le_rec
(n:nat)(x0:nat)(le n x0) -> (C:nat -> Prop)(C:n) -> ((m:nat)
  (le n m) /\ (C m) -> (C (S m))) -> (C x0)

```

We start proving the lemma by tactics theorem prover by the command Goal

```

->Goal (i j:nat) (le i j) -> (le (S i) (S j))

```

We make use of intro tactic below

```

->By intros

```

```

1 subgoal

```

```

  (le (S i) (S j))
  === == == == == =====
  H : (le i j)
  j : nat
  i : nat

```

```

->Pattern j

```

```

1 subgoal

```

```

  ([L:nat](le (S i) (S L)) j)
  === == == == == =====
  H : (le i j)
  j : nat

```

1 nat

Pattern j abstracts the present goal with the variable j

->Resolve H

2 subgoals

(le (S i) (S i))

=====

H (le i j)

j nat

nat

subgoal 2 is

(m nat)(le (S i) (S m))->(le (S i) (S (S m)))

->Resolve le_n

1 subgoal

(m nat)(le (S i) (S m))->(le (S i) (S (S m)))

=====

H (le i j)

j nat

i nat

->By intros

1 subgoal

(le (S i) (S (S m)))

=====

H0 (le (S i) (S m))

m nat

H (le j)

j nat

i nat

->Resolve le_S

1 subgoal

(le (S i) (S m))

=====

HO (le (S i) (S m))

m nat

H (le i j)

j nat

i nat

->By assumption

goal proved

The command Show_proof shows the current completed proof

->Show_proof

[i nat][j nat][H (le i j)][H [L nat](le (S i) (S L)) (le_n (S i))

[m nat][HO (le (S i) (S m))](le_S (S i) (S m) HO))

Save command saves the theorem so far proved for the rest of the session

->Save le_means_succ_le

le_means_succ_le is defined

The command Inspect num prints the last num items

->Inspect 2

le_means_succ_le (i nat)(j nat)(le i j)->(le (S i) (S j))

le_rec (n nat)(x0 nat)(le n x0)->(C nat->Prop)

(C n)->(C m nat)(le n m)/\ (C m)->(C (S m)))->(C x0)

That ends the session with the system

The command : *Reset After <ident fier>* resets the system to the state it was in after writing the identifier

The command : *Drop* completes the Vernacular mode and enters into the CAML mode The command *quit()* exits the system from the CAML mode

A112199

Date Slip

This book is to be returned on the
date last stamped

--	--

CSE-1991-M-NAR-PRO